VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

EXPERIMENTÁLNÍ SOFTWAROVÝ HUDEBNÍ NÁSTROJ, ZALOŽENÝ NA GRANULÁRNÍ SYNTÉZE A JEJÍ PROSTOROVÉ DISTRIBUCI

EXPERIMENTAL SOFTWARE MUSICAL INSTRUMENT BASED ON GRANULAR SYNTHESIS AND ITS SPATIAL DISTRIBUTION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Tomáš Bateško**
AUTHOR

VEDOUCÍ PRÁCE — **doc. Ing. MgA. Mgr. Dan Dlouhý, Ph.D.**
SUPERVISOR

BRNO 2019

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní obor **Audio inženýrství**
Ústav telekomunikací

Student: Tomáš Bateško ***ID:*** 195801
Ročník: 3 ***Akademický rok:*** 2018/19

NÁZEV TÉMATU:

Experimentální softwarový hudební nástroj, založený na granulární syntéze a její prostorové distribuci

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je kompletně naprogramovat experimentální elektroakustický softwarový hudební nástroj; jeho experimentálnost spočívá v netradičním způsobu řízení parametrů granulární syntézy a její prostorové distribuce.

DOPORUČENÁ LITERATURA:

[1] PUCKETTE, M. Theory and Techniques of Electronic Music, 2006. 337 s. online: http://msp.ucsd.edu/techniques.htm

[2] FORRÓ D. Svět MIDI. Grada, Praha, 1997. 375s. ISBN 80-7169-412-6.

Termín zadání: 1.2.2019 ***Termín odevzdání:*** 27.5.2019

Vedoucí práce: doc. Ing. MgA. Mgr. Dan Dlouhý, Ph.D.
Konzultant:

prof. Ing. Jiří Mišurec, CSc.
předseda oborové rady

Abstrakt

Cieľom tejto práce je vytvoriť experimentálny softwarový nástroj založený na granulárnej syntéze a konvolúcii schopný priestorovej distribúcie jednotlivých zŕn zvuku s možnosťou ich ovládania v reálnom čase súbežne viacerými užívateľmi po sieti. Zvukový zdroj je tvorený v jazyku Pure Data a grafické ovládanie pomocou webových technológii ako Node.js a p5.js.

Kľúčové slová

Granulárna syntéza, Pure Data, viackanálový priestorový zvuk, viac užívateľské rozhranie, webové technológie

Abstract

The goal of this thesis is to create experimental software instrument based on granular synthesis and convolution, capable of spatial distribution of individual grains and ability to control them in real-time over network by multiple users simultaneously. Sound source is created in Pure Data and graphical interface was built using web technologies such as Node.js and p5.js

Keywords

Granular synthesis, Pure Data, multichannel spatial audio, multiuser interface, web technologies

Bibliografická citácia:

BATEŠKO, Tomáš. *Experimentální softwarový hudební nástroj, založený na granulární syntéze a její prostorové distribuci*. Brno, 2019. Dostupné také z: https://www.vutbr.cz/studenti/zav-prace/detail/118120. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce Dan Dlouhý.

Prehlásenie

Prohlašuji, že svou bakalářskou práci na téma *Experimentální softwarový hudební nástroj založený na granulární syntéze a její prostorové distribuci* jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brne dňa: **27. mája 2019** ……

podpis autora

Poďakovanie

Ďakujem vedúcemu bakalárskej práce doc. Ing. MgA. Mgr. Danovi Dlouhému, PhD., za odbornú pomoc a cenné rady pri tvorbe a spracovaní tejto práce.

V Brne dňa: **27. mája 2019** ……

podpis autora

Obsah

Zoznam znakov a skratiek

Skratky:

Pd	…	Programovací jazyk Pure Data
Patch	...	Súbor jazyka Pure Data
Subpatch	...	podzložka v súbore jazyka Pure Data

Zoznam obrázkov

1. ÚVOD

Cieľom tejto práce je vytvoriť experimentálny softwarový hudobný nástroj využívajúci prvky granulárnej syntézy obohatenej o konvolúciu spolu s priestorovým viackanálovým zvukom a multimediálnym cross-platformovým viac-užívateľským ovládaním.

Užívatelia budú schopní v reálnom čase vytvárať a ovládať zvukové plochy (granulárne oblaky) tvorené z množstva veľmi krátkych zvukových úsekov z nahrávok alebo zvukových súborov klasickými parametrami z prostredia Pd. Užívatelia budú taktiež môcť priamo generovať jednotlivé granule pomocou ovládacích skriptov cez webové rozhranie v reálnom čase s možnosťou presného smerovania polohy jednotlivých čiastočiek zvukového oblaku alebo konvolučného dozvuku v priestore.

Zvuková časť nástroja je tvorená v grafickom programovacom jazyku Pure Data (Pd). Ovládanie nástroja je realizované z časti pomocou grafického rozhrania Pd a pomocou webových technológií ako Node.js a p5.js, ktoré sprístupňujú kompatibilitu väčšiny zariadení a ich periférií dnešnej doby. Pre viac užívateľské ovládanie som volil hierarchiu Client-Server, kde viacero užívateľov zo svojich zariadení komunikuje so serverom po lokálnej sieti a tak ovláda hudobný nástroj.

Tento projekt kombinuje kreatívne programovanie a tvorbu umelého zvuku a ich komunikáciu pomocou webových technológii. Skúma vplyv času a priestorového rozloženia zvukových prvkov krátkych intervalov v rôznych hustotách na ich výslednú tonalitu a ich vnímanie ľudským sluchovým aparátom. Zároveň sa snaží poukázať na prístupnosť informácii dnešnej doby tým, že využíva open-source zdroje na tvorbu multimediálneho diela.

2. TEÓRIA

Technológia dnešnej doby sa rýchlymi krokmi posúva napred a spolu s ňou vznikajú nové možnosti tvorby zvuku ako nikdy predtým. Hardvér bežného počítača alebo smartfónu dnes disponuje extrémne výkonnými procesormi, zvukovými prevodníkmi a perifériami, ktoré kedysi neboli predstaviteľné. Softvérová prístupnosť a kompatibilita zariadení taktiež nebola nikdy bližšie. Programovanie, siete a interaktivita je dnes prístupná každému. Tvoriť digitálne môže dnes ktokoľvek, ak má o to záujem. Internet je plný kreatívnych ľudí a komunít, ktoré inšpirujú a pomáhajú v riešení problémov. Všetka táto technika čaká na to, aby bola využitá novými kreatívnymi spôsobmi. Pre tento projekt s cieľom vytvoriť ako experimentálny zvuk, tak experimentálne ovládanie, som hľadal vhodné princípy a prístupné technológie, ktorými by bolo možné vytvoriť takýto nástroj.

2.1 Syntéza zvuku

Tvorba umelého zvuku alebo syntéza, pod týmto termínom je chápané spájanie častí, ktoré vytvárajú celok. Jeho význam stojí na tom, že je to kreatívny proces a nie len náhodné spájanie častí.

Syntezátor je teda prístroj alebo systém, ktorý umelo generuje úplne nový výstup. Existuje množstvo syntezátorov, ktoré sa odlišujú formou svojho výstupu a procesom jeho tvorby ako napríklad: syntezátor farieb, syntezátor textúr alebo zvukový syntezátor. Zvuková syntéza je potom proces umelého vytvárania zvuku s novou kvalitou. Tento proces môže byť zakaždým značne odlišný s rôznym výstupným produktom. Môže generovať zvuk elektronicky alebo elektro-mechanicky. Môže k tomu používať matematiku, fyziku, dokonca aj biológiu, spája dokopy umenie a vedu v hudobnú šikovnosť a technickú zdatnosť. [1]

Existuje množstvo typov syntéz tvorby zvuku. Je možno tieto typy rozdeliť do dvoch hlavných kategórii, a to **analógová syntéza** a **digitálna syntéza**. V následných podkapitolách predstavím historicky základné typy syntéz.

2.1.1 Aditívna syntéza

Princípom tejto syntézy je sčítavanie veľkého počtu sínusových priebehov s rôznymi frekvenciami a pomermi hlasitostí. Takto vzniká výsledná farba zvuku. Náročnosť u tejto syntézy je ovládanie veľkého množstva priebehov. [1]

2.1.2 Subtraktívna syntéza

Vychádza z princípu odčítavania z harmonicky bohatého signálu. Pomocou filtrov odstraňuje časti harmonického obsahu vstupného signálu. Vstupným signálom je typicky obdĺžnik, štvorec, trojuholník, pila alebo rôzne druhy šumov. [1]

2.1.3 Wavetable syntéza

Syntéza je zložená na cyklickom prehrávaní predom pripravených priebehov uložených v tabuľkách (wavetable). Tieto okná môže dynamicky meniť počas prehrávania [1].

2.1.4 FM a AM syntéza

Ide o modulačné syntézy. V prípade AM je signál amplitúdovo modulovaný iným signálom. So zvyšujúcou sa frekvenciou modulácie vznikajú postranné pásma a vznikajú nové zložky spektra.

V prípade FM je signál frekvenčne modulovaný jedným alebo viacerými signálmi. Takouto moduláciou vzniká veľké množstvo javov ako zrkadlenie frekvencii, otáčanie fáz a odčítanie jednotlivých vĺn, takto sú produkované veľké množstvá spektier. [1]

2.1.5 Samplovanie

Je vylepšenou verziou wavetable syntézy. Na rozdiel od používania krátkych priebehov vĺn, používa dlhšie vzorky zvukov, tie následne prehráva so zmenami tónu, dĺžky a pod. [1]

2.2 Granulárna syntéza

Granulárna syntéza je typ syntézy, ktorý spočíva vo využití mikroskopicky malých zvukových častí (typicky 1 – 100 ms), ktoré nazývame grains, zrná alebo granule, z ľubovoľného zvukového zdroja. Tie sú následne prehrávané v rýchlych intervaloch na hranici medzi vnímaním pulzu a kontinuálneho tónu. Takto tvorené zvuky vynikajú rozmanitosťou svojho spektra.

Medzi prvé zmienky o granulárnej syntéze patrí koncept maďarsko-britského fyzika a nositeľa Nobelovej ceny, Dennisa Gabora (1900-1979). Koncept spočíval v myšlienke, že každý tón je možné rozložiť na množstvo malých segmentov meniacich sa v čase. Na rozdiel Fourierovej nekonečnej rady, sa Gabor zameriaval na veľmi malé čiastočky zvuku, takzvané akustické kvantá a na ich psychoakustické vnímanie.

Jeden z prvých skladateľov zaoberajúcich sa a komponujúcich s mikrozvukom bol Iannis Xenakis[1] (1922-2001). Vo svojich kompozíciách experimentoval generovaním zvukov analógovými generátormi a skladaním krátkych zvukových ústrižkov na magnetickej páske. Medzi veľkých inovátorov v tejto oblasti určite patrí Curtis Roads[2] (*1951). Inšpirovaný Xenakisom, viedol výskum, tvoril zvuk granulárnymi metódami na počítači, kažkdý grain manuálne, pomocou dierovacích štítkov a neskôr implementoval tieto metódy do programovacieho jazyka C. Prvá implementácia digitálnej granulárnej syntézy v reálnom čase bol uskutočnená Barrym Truaxom[3] (*1947) v roku 1986.[1][2]

2.2.1 Časová škála

Vnímanie zvuku v ľudskom sluchovom aparáte je zložitý proces s mnohými premennými. Od zdroja zvuku, cez prostredie, ktorým sa šíri, po príchod do ucha poslucháča a následná premena energie na sluchový vnem. V tejto časti sa zameriavam na vplyv časového intervalu pôsobenia zvuku na jeho vnímanie Rozdelenie týchto časových intervalov bolo skvelo popísané v knihe Custisa Roadsa Microsound [2]. V nej delí časové úseky takto:

[1] Iannis Xenakis – grécko-francúzsky skladateľ, hudobný teoretik, architekt a stavebný inžinier
[2] Curtis Roads – americký skladateľ, autor, učiteľ a programátor
[3] Barry Tuax – kanadský skladateľ a učiteľ

- **Nekonečno (*Infinite*)**

  Nekonečne dlhý, matematicky ideálny interval ∞

- **Supra (*Supra*)**

  Čas prekračujúci hudobnú kompozíciu, typicky mesiace, roky, dni storočia, tisícročia.

- **Makro (*Macro*)**

  Doba trvania kompozície, typicky minúty, hodiny v extrémnych prípadoch dni.

- **Meso (*Meso*)**

  Rozdelenie kompozície do úsekov, zvukových objektov, typicky minúty a sekundy.

- **Zvukový objekt (*Sound object*)**

  Základná jednotka v hudbe – nota a komplexné mutácie zvukovej udalosti. Typicky sekunda až niekoľko sekúnd.

- **Mikro (*Micro*)**

  Zvukové čiastočky na mierke, ktorá hraničí s prahom vnímania zvuku, typicky tisíciny sekundy.

- **Audio vzorky (*Sample*)**

  Základný prvok digitálneho audia, vzorka 44,1kHz (mikrosekundy).

- **Kratšie ako audio vzorky (*Subsample*)**

  Zmeny krátke na to aby boli úspešne zaznamenané (nanosekundy).

- **Nekonečne krátky okamih (*Infinitisimal*)**

  Blížiace sa nule, ideálne krátky matematický okamih, Diracov impulz, delta.

Zaujímavosťou granulárnej syntézy je, že jej stavebné bloky sú na mikro škále, no vo výslednom ponímaní sú vnímané na škále zvukových objektov siahajúcich až na makro škálu. [2]

2.2.2 Rozdelenie a parametre

Hlavné druhy sa dajú rozdeliť do šiestich typov podľa organizácie tvorby granúl:

- **Matice a obrazy na časovo frekvenčnej doméne**

  Založené na Gaborovej matici tvorenej jeho formou analýzy.

- **Synchrónne prekrývajúce sa prúdy**

  Technika na vytváranie tonálnych zvukov. Začína analýzou spektra a následnom vytvorení rady filtrov, ktoré sa v čase menia a sú generované v presných okamihoch, aby sa navzájom prekrývali.

- **Synchrónne a kvázi-synchrónne prúdy**

  Zvuk vzniká z jedného alebo viacerých prúdov granúl. Jedna granula nasleduje druhú s určitým nemenným oneskorením. Synchrónnosť znamená, že zrná sú vytvárané v presných intervaloch. Hustotu zrna (*Density*) možno ovplyvniť rýchlosťou tvorby zrna za jednotku času (*grains per second*). Pri vyšších hustotách vzniká kontinuálny tón.

  Kvázi-synchrónne sú rozdielne v tom, že granule sú tvorené v nerovných intervaloch s prvkami náhodnosti. V tomto prípade záleží na veľkosti zrna (*grain size*) lebo môže vznikať prekrývanie alebo v opačnom prípade tiché momenty.

- **Asynchrónne oblaky**

  Opúšťa koncept lineárneho toku granúl a namiesto toho ich rozmiestňuje po určitú dobu popisovanú v časovo-frekvenčnej doméne. Takéto oblasti nazývame oblaky. Jednotlivé prvky sú ovládané alebo náhodné. Popis parametrov je abstraktnejší a zahrňuje dĺžku oblaku, veľkosť granúl, hustota granúl, frekvenčné centrum oblaku, amplitúdu a priestorové rozloženie.

- **Fyzikálne a abstraktné modely**

  Začína matematickým opisom akustického produktu alebo udalosti a snaží sa ho napodobniť. Napríklad práskanie ohňa, tok vody a pod.

- **Granulácia zvuku**

  Aplikácia granulárnych metód opísaných vyššie na navzorkované zvuky.[2]

2.2.3 Obálka zrna

Prehrávanú granulu je nutné amplitúdovo modulovať, aby nevznikali preskoky nenulových bodov na jej okrajoch a tým širokospektrálne hluky.

Na odstránenie týchto javov sa používa amplitúdová moduláciu okienkovou funkciou, ktorou vytvárame obálku granule. Takto sa vyhneme nenulovým preskokom a vyššie spomínaným artefaktom.

Používané okná môžu byť rôzne od jednoduchého rampového signálu až po zložité exponenciálne krivky. Každá okienková funkcia má svoje spektrálne vlastnosti a preto má výber okna vplyv na výsledný zvuk. Platí, že hranatejšie okná tvoria viac harmonických zložiek v spektre. Okná s jemnými hranami tvoria užšie. Každopádne sa táto modulácia viac, či menej stále prejaví vo výslednom spektre zvuku.[2]

2.3 Konvolúcia

Je matematickou operáciou dvoch funkcii, ktorej výstupom je tretia funkcia určená vzorcom (Rovnica 1). Konvolúcia má široké využitie v spracovaní signálov a je používaná na popis prenosových funkcii LTI (lineárny časovo invariantný) systémov. Prenosovú funkciu získame privedením jednotkového impulzu na vstup tohto systému. Čo znamená, že ak viem aký je signál na vstupe systému a poznám jeho prenosovú funkciu, teda impulznú odozvu, dokážem konvolúciou daný systém simulovať.

$$y(t) = h(t) * x(t) = \int_{-\infty}^{\infty} h(\tau)x(t-\tau)\,d\tau$$

Rovnica 1 Konvolúcia

Táto vlastnosť konvolúcie má časté využitie v simulácii dozvukov priestoru. Ak si predstavím priestor ako LTI systém, analogicky viem zistiť jeho prenosovú funkciu privedením jednotkového impulzu. Takto zaznamenanou impulznou odozvou dokážem konvolúciou pridávať dozvuk tejto miestnosti signálom. [12]

Možností ako vypočítať konvolúciu v číslicovej doméne je mnoho, no pre svoju rýchlosť vyniká algoritmus pomocou FFT (rýchla fourierová transformácia). Signály sú po blokoch prevedené z časovej domény do frekvenčnej a následne sú ich spektrá

násobené. Výsledné spektrum je prevedené do časovej domény pomocou IFFT (spätná rýchla fourierová transformácia). Tento postup odkrýva zaujímavý pohľad na konvolúciu zo strany tej, že konvolúcia je vlastne filtrom. [12] Rola konvolúcie v tejto práci spočíva z časti v simulácii priezoru no atypickými impulznými odozvami sa presadzuje ako filter.

2.4 Priestorový zvuk

Systémy reprodukcie zvuku sú v dnešnej dobe skoro všade. Podľa počtu kanálov ich môžeme rozdeliť na jednokanálové (mono), dvojkanálové (stereo) a viackanálové (surround sound). Ich vývoj postupoval od mona cez stereo až ku moderným surround systémom, za účelom vernejšej reprodukcie zdroja zvuku v priestore. V minulosti boli pokusy o verné zachytenie zvuku komplexným snímaním, nevýhodou bolo extrémne množstvo kanálov. Cieľom vývoja bolo vytvorenie metódy, ktorá by poskytovala ilúziu, ktorá by zredukovala počet kanálov. [3]

2.4.1 Ambisonia

Je matematický systém pre spracovanie signálov, ktorý sa snaží zachytiť a reprodukovať informáciu získanú z trojdimenzionálneho zvukového poľa. Je to proces kódovania a dekódovania signálu. Takýto signál môže byť získaný z reálneho zvukového prostredia špeciálnou metódou alebo ľubovoľný monofónny signál.

Kódovanie je proces začlenenia smeru pozície a vzdialenosti zvukového zdroja do prenosových kanálov. Ambisonia prvého rádu používa 4 prenosové kanály. Takto zakódované dáta nazývame B-Formát. Na rozdiel od bežného surround systému (napr. 5.1) prenosové kanály nemajú nič spoločné s reproduktormi, iba prenášajú informácie o danom zdroji a jeho pozícii v poly. Kódovanie B-formát prvého rádu má tieto koeficienty W - intenzita signálu, X – signál v ose x (hĺbka), Y – signál s ose y (šírka), Z – signál v ose z (výška).

Dekódovanie je proces získavania zvukových signálov, ktoré sú určené pre reprodukciu v danom reproduktorovom poly, z prenosových kanálov. Najprv je preto nutné určiť aké reproduktorové pole máme k dispozícii a následne mu pridať vhodný dekódovací proces.

Pomocou rovníc (Rovnica 2) je možné zakódovať ľubovoľný jednokanálový (mono) signál do B-formátu a tým určiť jeho pozíciu v zvukovom poly. Po zadaní φ je azimutu a ε elevácie sa výsledky týchto rovníc vynásobia so vstupným signálom a sú privedené na výstupy. [3]

$$W = 1 \div \sqrt{2}$$
$$X = \cos\varphi \cos\varepsilon$$
$$Y = \sin\varphi \cos\varepsilon$$
$$Z = \sin\varepsilon$$

Rovnica 2 Výpočet koeficientov pre B-Format

2.5 Pure Data

Pure Data (Pd) je open-source grafický programovací jazyk pre tvorbu zvuku a multimédií v reálnom čase. Vytvoril ho v rokoch 1995-1996 Miller Puckette, ktorý ho dodnes vyvíja a distribuuje jeho verziu s názvom Pd-Vanilla. Funguje na mnohých, dnes populárnych platformách, ako napríklad Windows, GNU/Linux, Mac OS X, iOS či Android. Značnou výhodou Pd je jeho otvorenosť a portabilita, za ktorou stojí aktívna komunita užívateľov a vývojárov vytvárajúcich nové rozšírenia.

Pd sa vyznačuje grafickým prostredím štýlu patcher, v ktorom užívateľ spája funkčné objekty pomocou čiar a tým prepája ich funkcionalitu. Pd zdieľa značnú podobnosť v prostredí s programom Max/MSP, na ktorom vývoji sa Puckette podieľal v jeho začiatkoch.

S rozšíreniami, ako napríklad GEM (Graphics Enviroment for Multimedia) je možné tvoriť a manipulovať video, OpenGL grafiku, obrázky v reálnom čase. Sieťovú komunikáciu po LAN alebo internete je natívne podporovaná a ako sieťový protokol používa FUDI. Podporuje protokoly MIDI a OSC. Pomocou knižnice LibPd je možné zapuzdrovať Pd patche do natívnych programov pre Android, iOS a vďaka JavaScriptovej knižnici WebPd je možné patche spúšťať vo web prehliadačoch. [4]

2.5.1 Popis prostredia

Dokumenty vytvorené pomocou Pd sa nazývajú *patche*. Obsah patchu pozostáva z objektov niekoľkých typov pospájaných dokopy. Objekty sa líšia svojimi okrajmi

a každý je interpretovaný v Pd inak, (Obr. 2-1). Za každým z objektov sa skrýva kód v programovacom jazyku C, čo robí z Pd veľmi efektívne a schopné vývojové prostredie. Nad kompilovanými textovými zvukovými programovacími jazykmi preto vyniká v intuitívnom prototypovaní a zmenami v reálnom čase. Hlavné typy objektov:

- **Objekt** *(object box)*

  Majú obdĺžnikové telo, interpretujú vpísaný text pri ich vytvorení. Môžu v sebe obsahovať rôzne triedy objektov – oscilátory, obálkové generátory, a ďalšie DSP moduly. V tomto prípade je to objekt *osc~*, ktorý na svojom výstupe generuje harmonický signál o danej frekvencii s amplitúdou 1. Objekty zakončené znakom ~ pracujú na vzorkovacej frekvencii DSP.

- **Správa** (*message box*)

  Rohy objektu v tvare vlajky. Interpretujú sa ako text, vždy ak je objekt aktivovaný (vstupnou správou alebo myšou).

- **Číslo** (*number box*)

  Je špecifický typ grafického (GUI) objektu. Medzi ďalšie patria tlačidlá (*buttons*), prepínače (t*oggles*) a iné. Tieto GUI objekty môžu meniť svoj tvar a funkčnosť podľa ich typu. [4][5]

Obr. 2-1 Základné objekty

2.5.2 Enkapsulácia v Pd

Pd je možnosť enkapsulácie, ktorá dovoľuje tvoriť patche, ktoré môžu byť znova využívané. Enkapsulácia v Pd má dve formy, a tie sú *subpatch* a *abstrakcia*.

Vytvorením objektu, ktorý má pred názvom "pd" vznikne takzvaný *subpatch*. Je ukladaný ako súčasť rodičovského patchu. Možno vytvoriť viacero jeho kópií a každú z nich je možné individuálne upravovať.

Druhou možnosťou je *abstrakcia*. Pri vytváraní objektu sa v mene volá už existujúci Pd patch (súbor *.pd*). To otvorí daný súbor v subpatchy. Je možne

ľubovoľne kopírovať abstrakcie s tým rozdielom, že zmena v jednej abstrakcii sa prejaví vo všetkých. [4]

2.5.3 Knižnice

V Pd je možné inštalovať externé knižnice a využívať tak objekty vytvorené inými používateľmi. Inštalovať knižnice je možné manuálne, kompiláciou a následne pridávaním cesty k danému súboru knižnice.

Vo verzii *Pd Vanilla,* ktorá je spravovaná Millerom Pucketom, je možné knižnice inštalovať pomocou frameworku s názvom *Deken*. Ten po vpísaní názvu knižnice prehľadáva svoj verejný GitHub repozitár a po jeho nájdení ju nainštaluje[5]. Výhoda používania knižníc oproti vytvoreniu potrebných funkcii v Pd je ich efektivita, keďže sú napísané ako objekty v programovacom jazyku C. V ďalších podkapitolách sú popísané externé knižnice dôležité pre túto prácu.

2.5.3.1 mrpeach

Jedná sa o knižnicu, ktorú vytvoril v rokoch 2006-2015 Martin Peach, a bola vydaná pod licenciou GNU ako slobodný softvér. Disponuje veľkým množstvom objektov na spracovanie sieťových komunikácii protokolov ako UDP, OSC, HTTP, FUDI, binárne operátory, systém celulárnych automatov (Conway´s game of life) a mnoho ďalších. V tejto práci sú využívané hlavne objekty pre čítanie, rozbaľovanie a triedenie OSC komunikácie.

2.5.3.2 grambilib~

Funkciou tejto knižnice je kódovanie jednokanálových signálov do formátu *Ambisonic B-format* a ich následné dekódovanie do potreného počtu a rozostavenia reproduktorov. Vytvoril ju Ricky Graham v roku 2006. Táto knižnica vyniká svojou jednoduchosťou definíciou priestorového rozloženia zvuku. Poskytuje kódovanie signálov (2.4.1) pomocou objektu *grambipan~*, až do tretieho rádu. Dekóduje *B-Format* signál pomocou objektu *grambipan~* , to v konfiguráciách mono, stereo, pent (5 reproduktorov horizontálne), hex (6 reproduktorov horizontálne), oct (8 reproduktorov horizontálne), cube (8 reproduktorov s 3d rozmiestnením, *Ambisonic Cube*). Obsahuje aj objekt pre manipuláciu zvukového zdroja *grambiman~* s argumentom *rotate, tilt, tumble, rotilt*, ovládaný signálovými vstupmi.

2.6 Komunikácia

V analógovej ére bola komunikácia medzi jednotlivými zariadeniami uskutočňovaná pomocou zmien napätí, pulzov a trigger signálmi, ktoré sú dodnes používane v určitých aplikáciách. Veľký rozvoj však nastal v 80-tych rokoch s príchodom protokolu MIDI.[1]

2.6.1 MIDI

Protokol MIDI (*Musical Instrument Digital Interface*) slúži na výmenu informácii medzi elektronickými hudobnými zariadeniami a počítačmi. Dáta, ktoré posiela sú hudobné informácie, ako napríklad stlačenie klávesy, pohyb potenciometrom, nie však zvuk ako taký. Je najrozšírenejším hudobným protokolom a je podporovaný drvivou väčšinou všetkých DAW.

Na prenos MIDI používa digitálnu sériovú komunikáciu, to znamená, že posiela dáta v binárnej forme po kábli. Prenos týchto dát je prenášaný prúdom a nie napätím. Fyzicky má vlastný 5 pinový port. Posiela rôzne typy správ ako správy o spojení zariadení, zmeny programu, notové udalosti, exkluzívne systémové správy a poskytuje MIDI 16 kanálov. [1]

2.6.2 OSC

Modernú alternatívu ku protokolu MIDI predstavuje protokol OSC. Open Sound Control (OSC) je open-source, prenosovo nezávislý, komunikačný protokol medzi počítačmi, zvukovými syntezátormi a inými multimediálnymi zariadeniami. Optimalizovaný na moderné sieťové technológie, prináša benefity sietí do sveta elektronických hudobných nástrojov.

Medzi hlavné výhody protokolu OSC patrí presnosť, flexibilita, jednoduchosť používania. Je ľahko implementovateľný, spoľahlivý a komunikuje po sieti.

Je implementovaný v mnohých programovacích jazykoch pre tvorbu zvuku ako napríklad Pure Data, Max/MSP, SuperCollider, Processing, Logic Pro, Csound, Juce, Nuendo.

OSC protokol posiela dva typy správ *OSC Message* a *OSC Bundle*. *OSC Message* pozostáva z adresy, typového tagu a z argumentov. *OSC Bundle* sú zložené OSC správy, ktorých efekt sa vykonáva súčasne. Prakticky sa od MIDI odlišuje tým,

že je ním možné dané správy lepšie formátovať. Je možné zmeniť ich názov a tým ich lepšie smerovať alebo meniť rozsah ich hodnôt, na rozdiel od rozsahu 0-127, ktorým disponuje protokol MIDI [6].

2.7 Zdieľané ovládanie/ Realtime Web

Vytvárať užívateľské rozhrania je možné mnoho spôsobmi. Tvoriť natívne aplikácie pre každú platformu zvlášť. Značnou výhodou je ich vysoký výkon a optimalizácia danej aplikácie. Značnými nevýhodami je, že každá platforma vyžaduje vývoj aplikácie samostatne. Tento proces je časovo náročný a zložitý na znalosti.

Ďalšia možnosť je pokúsiť sa o zjednotenie vývoja tvorením jednej webovej aplikácie kompatibilnej s viacero zariadeniami. Interaktívna HTML webstránka s JavaScript prvkami je zobrazovaná a funkčná v reálnom čase ovládateľná a na rôznych zariadeniach fungujúca bez zmeny. Preto sú nasledujúce technológie v dnešnej dobe často používané pre tvorbu webových aplikácii a taktiež v tomto projekte. [11]

2.7.1 Processing / p5.js

Processing je open-source programovací jazyk/grafická knižnica na tvorbu grafiky a takzvaný *„creative coding“* spájajúca umelcov a programátorov. Ako jadro používa programovací jazyk Java. Vytvorili ho študenti MIT Media LAB Casey Reas a Ben Fry v roku 2001.

Slúži na tvorbu vizuálneho dizajnu a svojou jednoduchosťou je ideálnym prostriedkom k výučbe programovania počítačovej grafiky a programovania ako takého. Jeho aktívna komunita prispieva množstvom návodov a stovkami knižníc uľahčujúce vizualizáciu dát, sieťovú komunikáciu, 3D grafiku a tvorbu Android či iOS aplikácii.[7][8]

P5.js je natívna JavaScript alternatíva k pôvodnému Proccesingu, ktorým je zároveň podporovaná. V celkovom syntaxe sa od pôvodného Processingu líši len minimálne, čiže užívatelia dokážu používať obe verzie podľa potreby bez väčších problémov. Cieľom p5.js je priblížiť programovanie umelcom, dizajnérom a učiteľom s možnosťou jeho využitia na webových stránkach.

Keďže p5.js je v podstate JavaScript knižnica, dá sa použiť v HTML volaním skriptu, s ktorým dokáže kooperovať. Okrem grafiky obsahuje veľa knižníc na zvuk, fyziku, 3D rendering pomocou WebGL, využitie mobilných zariadení a ich periférii ako gyroskop, akcelerometer, kamera a ďalšie. V rámci skriptu dokáže nielen vykresľovať dáta, ale aj komunikovať s jednotlivými HTML objektami na web stránke. Ďalšou značnou výhodou je kombinácia s inými JavaScript knižnicami a celkovo sieťovými technológiami. [9]

2.7.2 Matter.js

Je JavaScript engine na počítanie 2D fyziky objektov a ich interakcie. Obsahuje triedy pre jednotlivé typy objektov, svet, v ktorom sú vyvolané s možnosťou zmeny parametrov ako gravitácia, váha, odrazivosť a podobne. Ma vlastný render engine na vykresľovanie. Poskytuje užitočné dáta o kolíziách objektov a disponuje užívateľskou interakciou s nimi.

2.7.3 Node.js

Node.js je platforma postavená na JavaScript runtime prehliadača **Chrome**. Slúži na jednoduchú tvorbu rýchlych a škálovateľných webových aplikácii. Používa udalosťami riadený, asynchrónny, vstupy neblokujúci model, ktorý z neho robí efektívny systém s využitím pri tvorbe dátovo náročných a v reálnom čase pracujúcich aplikácii. Na vykonávanie svojich funkcii používa event callback systém a event listenerom, čo znamená, že čaká (event listener) a na vykonanie udalosti (event) a pri jej udiatí vykoná funkciu jej pridelenú (callback).

Node.js disponuje správcom svojich knižníc s názvom **npm** (*node package manager*), pomocou ktorého je možné sťahovanie a inštalovanie nových knižníc, potrebných pre danú webovú aplikáciu.

Veľkou výhodou je, že poskytuje serverový framework v programovacom jazyku JavaScript. Takto podporuje koncept JavasScript všade, čo znamená, že ako backend (serverová časť) aj frontend (užívateľská časť) aplikácie/stránky sú vytvorené pomocou neho. [10][11]

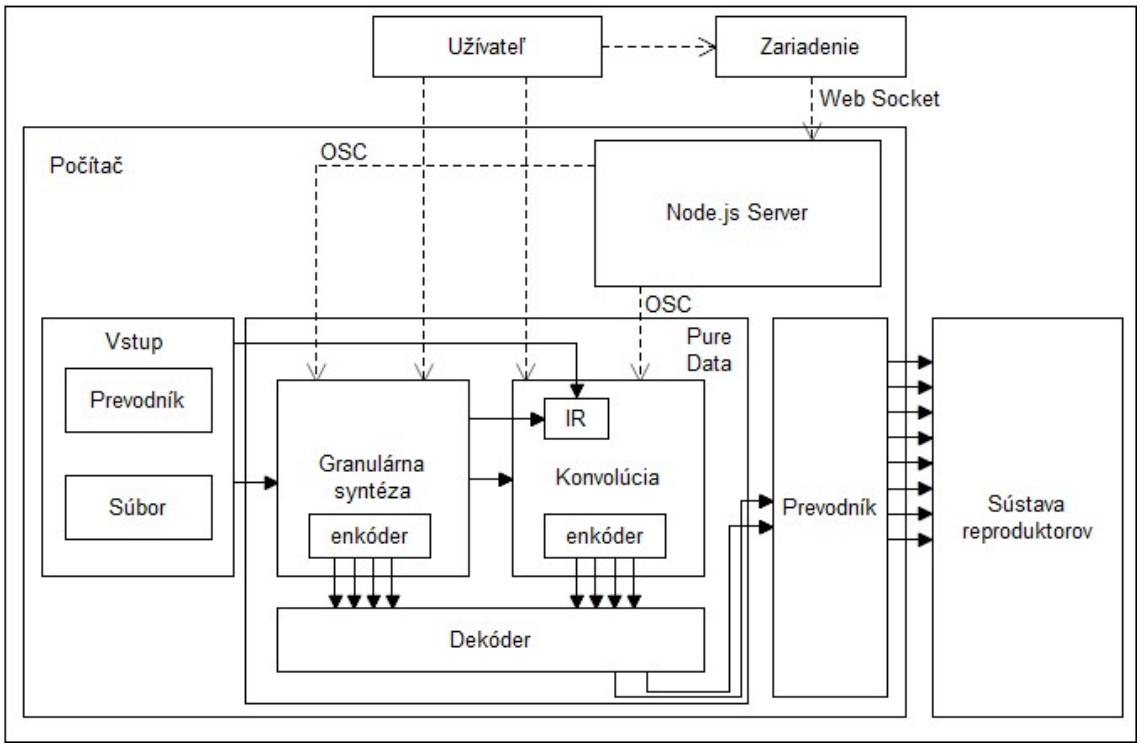
2.7.4 Express.js

Tento modul Node.js je framework slúžiaci ako vrstva na zjednodušenie ovládania spravovania a tvorby infraštruktúry serveru v rámci Node.js. Definuje port, zdieľanie súborov, zabezpečenie servera a pod. [11]

2.7.5 Socket.io

Je modul Node.js, ktorý slúži ako vrstva abstrakcie ku komunikačnej funkcii WebSocket implementovanej v HTML5. WebSocket je funkcia na obojsmernú komunikáciu medzi webovou stránkou/aplikáciou a serverom pomocou sieťového soketu. Socket.IO zjednodušuje prácu s WebSocket v prostredí Node.js. Je možné ho jednoducho nainštalovať pomocou **npm**. Jeho inicializáciou vytvárame komunikáciu medzi aplikáciou a serverom , ktorú môžeme potrebne spracovávať. [11]

3. NÁVRH



Obr. 3-1 Bloková schéma nástroja

Predstava je vytvoriť hudobný nástroj, kombinujúci známe prvky v oblasti zvukovej syntézy, multimédií a interaktívnej tvorby spôsobom, ktorý je efektívne využiteľný a zároveň poskytuje veľké možnosti experimentovania s interaktivitou

a generovaním zvuku v reálnom čase. Projekt je založený na open-source aplikáciách a ich možnostiach kreatívneho využitia.

Na blokovej schéme hudobného nástroja (Obr. 3-1), sú názorne zobrazené jednotlivé bloky a ich komunikácia s ostatnými blokmi. Užívateľ bude mať možnosť vložiť alebo nahrať zvukovú stopu, ktorá bude použitá na granulárnu syntézu. Následne bude tento zvuk spracovaný konvolučným algoritmom s premennou impulznou odozvou. Impulznú odozvu bude možné vkladať zo súboru, nahraním zo vstupu prevodníku alebo nahratím výstupu z granulárneho syntezátora.

Ovládanie základných prvkov granulárnej syntézy, jej rozloženia v priestore ako aj rozloženia konvolúcie v priestore, bude možné ako lokálne, z prostredia programovacieho jazyka Pd, tak aj vzdialene, viacero užívateľmi z ľubovoľných zariadení súčasne cez lokálnu sieť.

Predstavou je oddeliť zvukovú zložku od ovládacej a ich komunikáciu centralizovať cez server. Výsledné mixtúry spájať v určitých pomeroch na výstup do viackanálového výstupu. Zvukový výstup bude zakódovaný vo formáte *Ambisonics B-Format*, ktorý bude možné dekódovať do rôznych zoskupení reproduktorov. Využiť rôzne zdroje vhodné na ovládanie týchto prvkov a sprístupniť toto ovládanie na mnoho platform (Android, iOS, Windows, OSX).

4. REALIZÁCIA

4.1 Zvukový zdroj a interné ovládanie

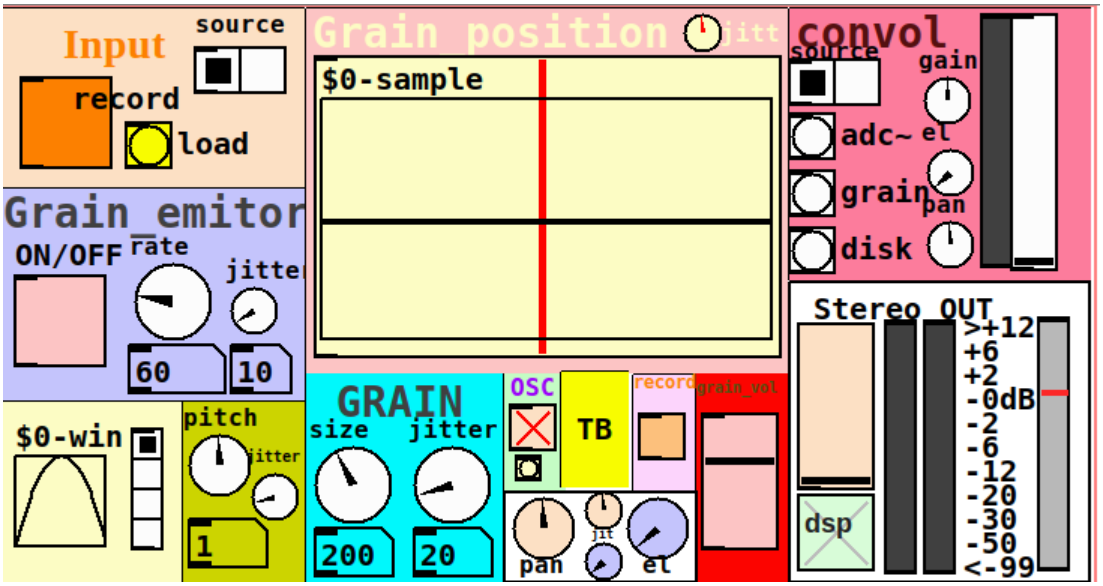
Hlavnou častou tejto práce je tvorenie zaujímavých priestorovo distribuovaných zvukových paliet, ktoré možno dynamicky ovládať v reálnom čase. Ovládať nástroj je možné vzdialene, ale aj lokálne priamo z patchu v Pd. Ovládanie z Pd ma väčšiu kontrolu nad parametrami ako napríklad načítanie zvukov a zvolenie prevodníkov na vstupy a výstupy. V tejto kapitola popíšem realizáciu princípu tvorby týchto zvukov v programovacom prostredí Pure Data.

4.1.1 Hlavné okno

Táto sekcia je hlavnou časťou syntezátora. Tvorí všetok zvuk a disponuje hlavným ovládacím rozhraním syntezátora v samotnom prostredí Pd, ktoré

je nadradené ovládacím skriptom. Každý subpatch ovláda alebo generuj určitý prvok granulárnej syntézy, jej priestorového rozloženia, konvolúcie, nahrávania výstupu, alebo ovláda externé vstupy a komunikáciu po sieti pomocou protokolu OSC. Jednotlivé bloky systému komunikujú medzi sebou internými správami a premennými, ktoré posielajú svoje hodnoty synchrónne. Synchronizácia udalostí je generovaná pomocou premennej $0-emitBang (4.1.10). Takéto synchronizovanie užívateľských vstupov je kľúčové pre odstránenie zvukových chýb a praskania, pretože nedovoľuje zmenu ovládacích premenných syntezátora pri každej zmene GUI objektu počas prehrávania. Užívateľ môže vykonávať zmeny s prvkami, no tie sa prejavia až pri ďalšom takte synchronizácie. Jednotlivé subpatche tvoria grafické ovládanie pomocou kalsických GUI objektov v Pd.

Funkčnosť a ovládanie jednotlivých prvkov detailne opíšem v následných podkapitolách. Na obrázku (Obr. 4-1) vidno celkové usporiadanie subpatchov a ich jednotlivé ovládacie komponenty na hlavnom okne. Po otvorení master patchu sa toto okno zobrazí.



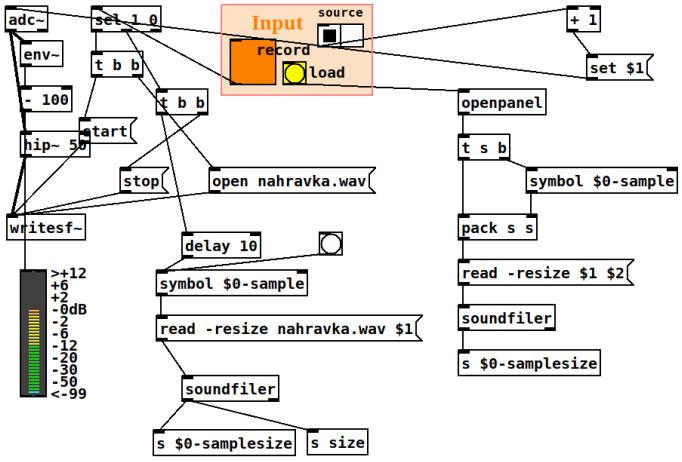
Obr. 4-1 pd main

4.1.2 pd input

V tomto subpatchi (Obr. 4-2) je možné nahrávať zvukový vstup z prevodníka zvoleného na zariadení pomocou objektu *adc~*. Je možné vybrať vstupný kanál daného prevodníka pomocou objektu *hradio*. V tomto prípade je možný výber medzi prvými dvoma kanálmi. Pomocou objektu *toggle* s názvom *record* sa spúšťa

a zastavuje zapisovanie zvoleného vstupu do súboru *sample.wav* na disk. Následne je tento súbor vložený do tabuľky *$0-sample*, z ktorej je ďalej používaný v programe. Ďalšou možnosťou je načítať, už existujúci zvukový súbor vo formáte *.wav* z disku. To realizujeme stlačením tlačidla *load*, ktorý vyvolá formulár, kde užívateľ zadá cestu k danému súboru a ten sa opäť zapíše do tabuľky *$0-sample*.

Tabuľka *$0-sample* dynamicky mení svoju veľkosť pri každej zmienenej zmene, čo je výhodou, ale zároveň aj nebezpečím. Treba dbať na vhodnú veľkosť súboru, pretože Pd nemá veľkostné obmedzenie pre súbory a mohla by nastať chyba. Vhodná veľkosť je myslená tak, že súbor by nemal presiahnuť veľkosť pamäte RAM na zariadení.



Obr. 4-2 pd input

4.1.3 pd brain

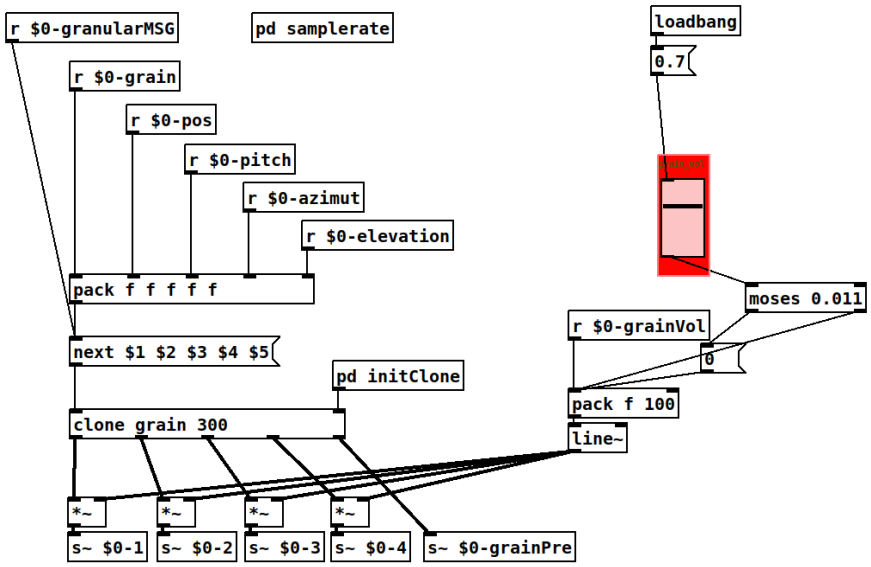
Úlohou tohto subpatchu (Obr. 4-3) je klonovanie abstrakcie *grain* (4.1.9) pomocou objektu *clone*, získavanie správ od ostatných častí patchu, spracovať ich a posielať ich jednotlivým klonom, aby mohli vykonať prehrávanie.

Prijíma samostatné správy o veľkosti, pozícii, výške tónu a priestorovej distribúcii od ostatných ovládacích subpatchov, ktoré zabaľuje pomocou objektu *pack* do správne formátovaného listu. Druhý spôsobom je prijatie už zabaleného listu z premennej s názvom *$0-granularMSG* zo subpatchu *oscStream* (4.1.12), ktorý je už formátovaný.

Oba tieto listy putujú do správy objektu *clone* so znením *„next $1 $2 $3 $4 $5”*, ktorá tieto dáta pošle ďalšiemu voľnému klonu abstrakcie. Tento postup umožňuje použitie všetkých správ v samostatnom klone objektu, čoho výsledkom je možnosť

súbežného ovládania zvuku priamo z prostredia Pd a zároveň aj z externých skriptov viacero užívateľmi po sieti (4.2.2). Každá z týchto správ je premenená na samostatný zvuk, ktorý je schopný prekrývania s ostatnými.

Zvuk tvorený klonmi je kódovaný *Ambisonic B-Format* a jeho štvorici kanálov je možné upraviť hlasitosť a následne sú posielané objektom *send~* do ďalšej vrstvy na dekódovanie a spracovanie.



Obr. 4-3 pd brain

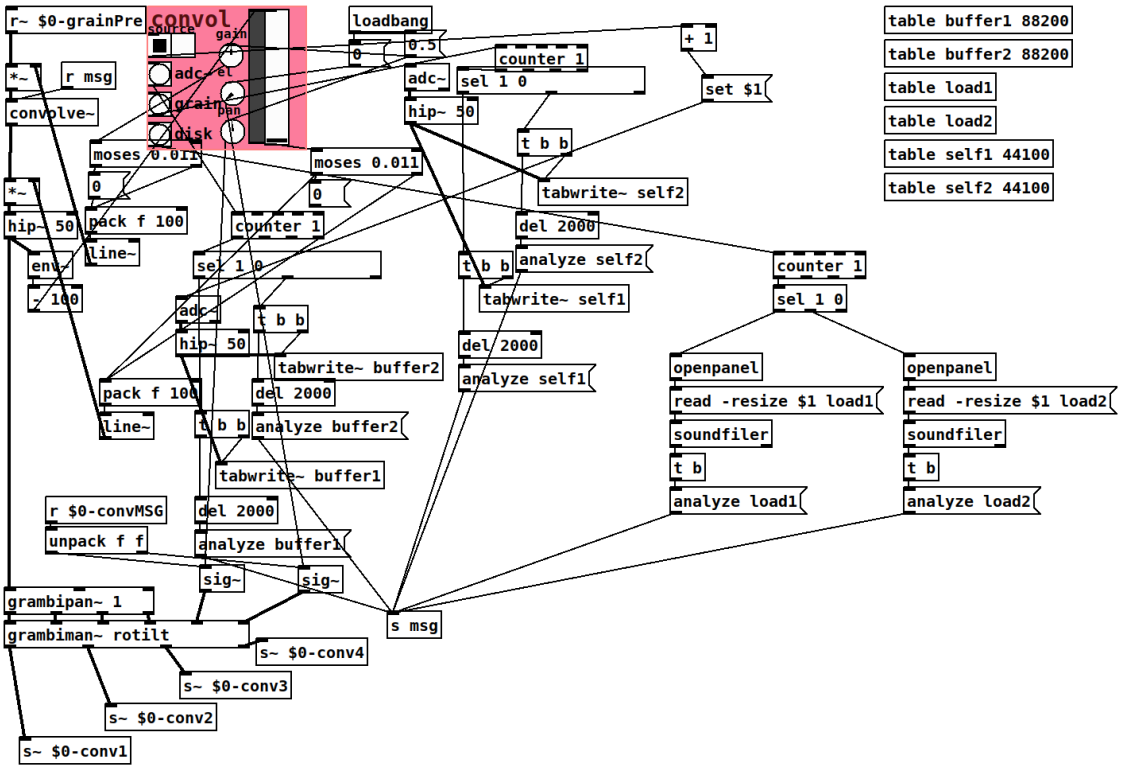
4.1.4 pd conv

Táto abstrakcia umožňuje vykonávanie konvolúcie signálu granulárnej syntézy s ľubovoľnou impulznou odozvou v reálnom čase. Takto vytvára priestorové dozvuky alebo experimentálne filtračné zvuky. Základom je objekt *convolve~*, ktorý vykoná konvolúciu vstupného bloku signálu s blokom impulznej odozvy zapísanej v tabuľke, ktorú má predom zadanú a analyzovanú. Signál konvoluje pomocou FFT blokov oboch signálov a následným násobením ich spektier navzájom. Následne vykoná spätnú FFT a takto získaný signál v časovej doméne privádza na svoj výstup (2.3). Impulznú odozvu je možné meniť v reálnom čase za inú s chvíľkovým výpadkom zvuku na výstupe tohto objektu. Výber impulznej odozvy je vykonaný pomocou správy do objektu *convolve~* o tom, ktorú tabuľku má analyzovať a použiť.

Ako impulznú odozvu je možné použiť záznam vstupu z prevodníku o dĺžke jednej sekundy stlačením tlačidla *adc~*, načítaný súbor impulznej odozvy z disku pomocou tlačidla *disk* alebo nahraním 2-sekundovej vzorky impulznej odozvy

z mono výstupu granulárneho syntezátora tlačidlom *grain~*. Impulzná odozva môže byť ľubovoľný zvuk.

Výstup z konvolúcie je taktiež kódovaný objektom *grambipan~*, ktorý ho prevedie z mono signálu na štvorkanálový *Ambisonic B-Format*. Jeho rozloženie v priestore je možné meniť manuálne potenciometrom *pan*, ktorý posúva signál v protismere hodinových ručičiek po horizontálnej ose, potenciometrom *el* ovládajúci zdvih signálu vertikálnej ose, podobne ako v subpatchy *pd pan* (4.1.7). Separátny kódovač dáva možnosť pohybovať konvolučným signálom nezávisle od zvuku granulárneho syntezátora v priestore. Prijíma aj vlastné OSC správy, takže je možné jeho pozíciu meniť z externých skriptov. Štvorica kódovaných signálov je posielaná na pomocou objektov *send~* do ďalších subpatchov na dekódovanie (4.1.8).



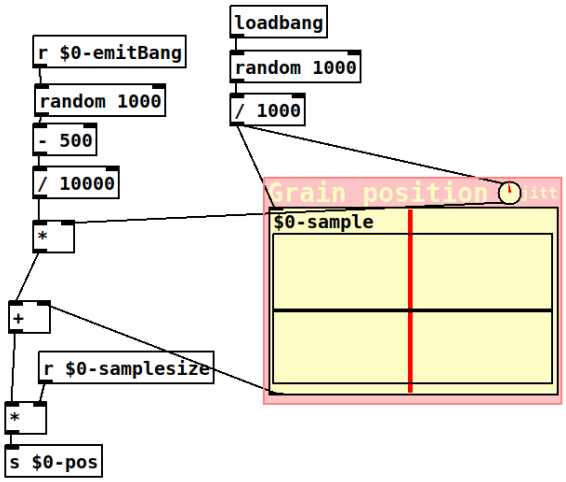
Obr. 4-4 pd conv

4.1.5 pd grain position

Funkciou tohto subpatchu (Obr. 4-5) je zadávanie začiatočnej (offset) pozície v zvukovom súbore, z ktorého sú prehrávané jednotlivé granule.

Grafické ovládanie je vytvorené objektom tabuľky *array* s názvom *$0-sample*, v ktorej je zapísaný zvukový súbor na granuláciu. Ten zároveň slúži ako zobrazenie

zvuku v časovej doméne. Nad tabuľkou je posadený objekt *vslider,* ktorého chodec je zobrazený červenou farbou. Jeho pohybom nad súborom užívateľ volí pozíciu offsetu. V pravom rohu sa nachádza objekt v *mknob* s názvom *jitt*, určujúci množstvo náhodne generovanej odchýlky od užívateľom zadanej pozície. Sčítaním týchto hodnôt získame hodnotu pozície, ktorá je posielaná ďalej v premennej *$0-pos*.

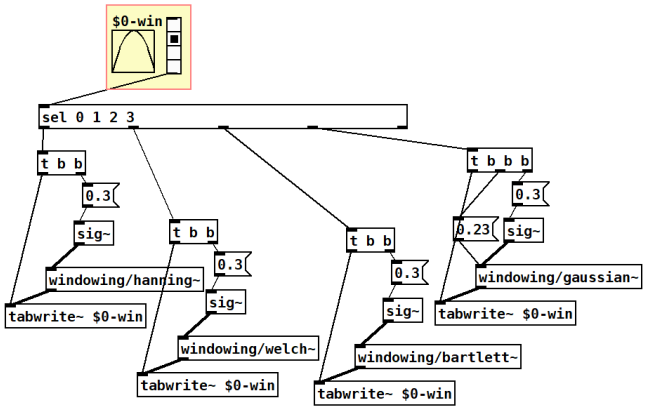


Obr. 4-5 pd grain position

4.1.6 pd window

V tomto subpatchi (Obr. 4-6) sa vytvára tabuľka s priebehom, ktorý je použitý ako obálka zrna (2.2.3). Dĺžka tohto priebehu je 64 samplov. Táto krátka dĺžka nevadí, pretože jednotlivé hodnoty z tabuľky sú čítane objektom *tabread4~* (4.1.9), ktorý medzi nimi interpoluje. Vertikálnym prepínačom je možné voliť medzi štyrmi priebehmi alebo do tabuľky ľubovoľný priebeh nakresliť. Priebehy sú vytvárané svojím vlastnými objektami z knižnice *windowing*. Priebehy sú následovné:

1. hanningove okno
2. welchovo okno
3. bartlett okno
4. gaussové okno



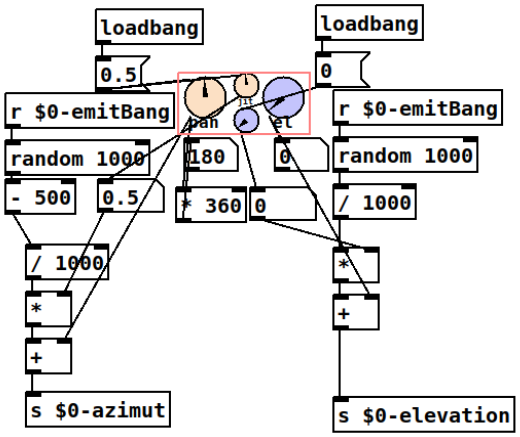
Obr. 4-6 pd window

4.1.7 pd pan

Tento subpatch (Obr. 4-7) disponuje ovládacími prvkami pre manuálne ovládanie priestorového rozloženia zvukov (4.1.8) nezávislého od externých vstupov.

Objektom *mknob* s názvom *pan* je možné smerovať prehrávane granule po horizontálnej osi od 0° po 360° v protismere hodinových ručičiek. Tento objekt funguje ako vo viac reproduktorových formáciách, tak aj v stereo rozložení. Jediným rozdielom medzi konfiguráciami je ten, že stredom stereo bázy je v prípade sterea 180°, kde obe reproduktory hrajú s rovnakou hlasitosťou. V prípade mnohokanálovej reprodukcie je to 0°. Vedľa v rovnakej farbe je taktiež *mknob* s názvom *jitt*, ktorý ovláda množstvo náhodne generovanej odchýlky od zadanej hodnoty.

Ďalšia dvojica objektov *mknob* s rovnakou farbou s názvami *el* a *jitt* ovláda podobne ako predošlá dvojica, zdvih zvuku v priestore a jeho množstvo náhodne generovanej odchýlky od zadanej hodnoty. Tento objekt funguje primárne pri dekódovaní do 3D rozloženia kanálov *Ambisonic Cube* (Obr. 4-9), no jeho zmeny počuť aj v stereu a iných horizontálnych viackanálových rozloženiach. V 2D sa prejaví ako celkový pokles hlasitosti. Užívateľom zadané hodnoty posiela do ďalších subpatchov v premenných *$0-azimut* a *$0-elevation*.



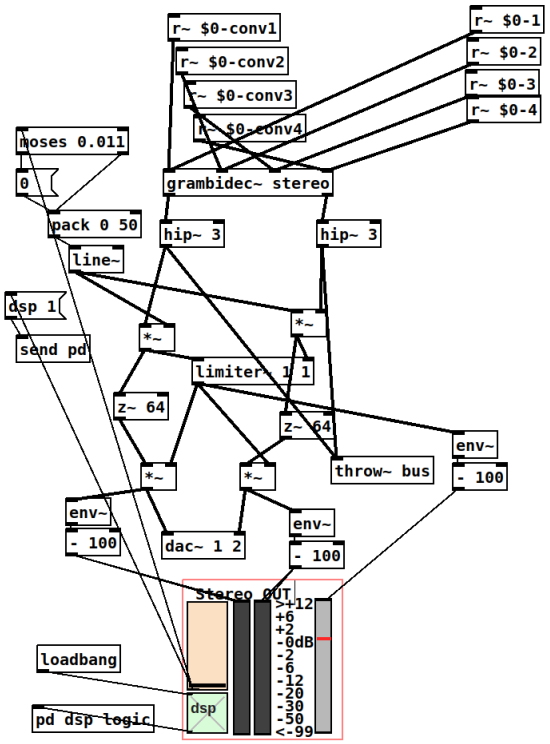
Obr. 4-7 pd pan

4.1.8 pd stereo/cube output

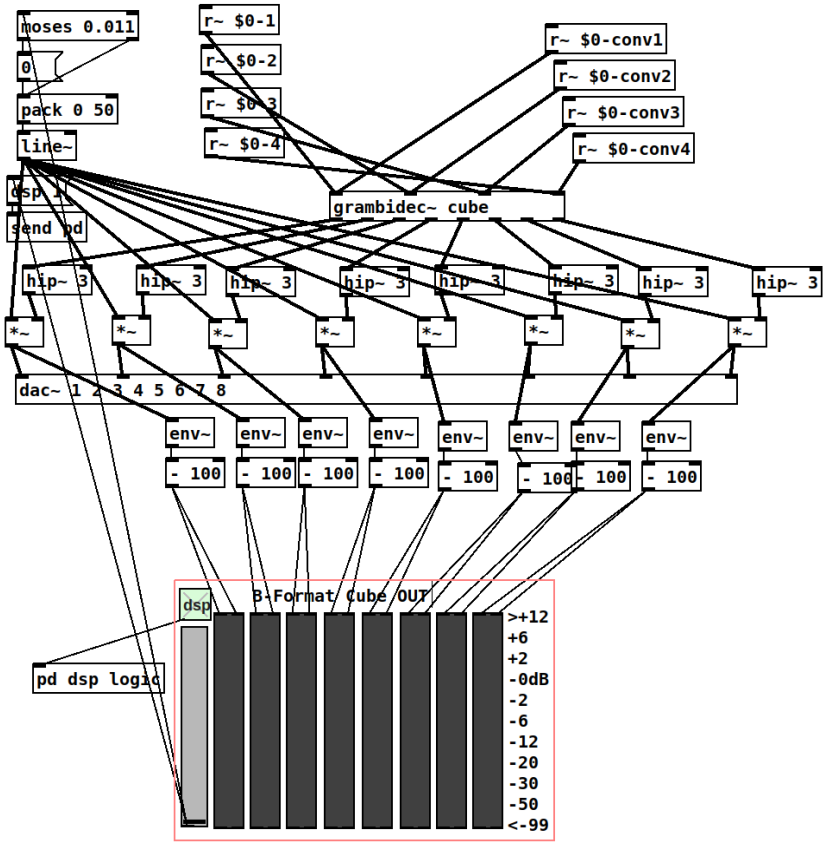
Telom týchto subpatchov je objekt *grambidec~* z knižnice *grambilib*, ktorý dekóduje štvoricu kanálov *Ambisonic B-Format* prijatých z ostatných subpatchov do potrebnej konfigurácie reproduktorov. V terajšom stave je *pripravená* dvojica dekódovaní a to stereo dekóder s limiterom (Obr. 4-8) a 8-kanálový dekóder do sústavy *Ambisonic Cube* (Obr. 4-9).

Do dekodérov sú privedené *Ambisonic B-Formát* výstupné kanály z klonov abstrakcie *grain* (4.1.9) a subpatchu *conv* (4.1.4). Tie sú spoločne dekódované do potrebného formátu reproduktorov. Amplitúda signálov jednotlivých kanálov je konvertovaná na správu objektom *env~* a zobrazovaná objektom *VU Meter*.

Výstupy kanálov sú filtrované filtrom horná priepusť o frekvencii 3 Hz kvôli jednosmernému offsetu a sú privedené na príslušné výstupy zvukovej karty pomocou objektu *dac~* s argumentom počtu signálov. Oba subpatche taktiež disponujú nastavením celkovej výstupnej hlasitosti pomocou objektu *vslider*. Je možné zapínanie a vypínanie DSP processingu zeleným tlačidlom *toggle*, čím možno ihneď vypnúť výstupný zvuk.



Obr. 4-8 pd stereoOut



Obr. 4-9 pd cubeOut

4.1.9 Abstrakcia grain

Táto abstrakcia (Obr. 4-10) je hlavným prehrávacím prvkom jednej granule zvuku. Je uložená ako samostatný *.pd* súbor, aby mohla byť použitá v objekte *clone* (4.1.3), ktorý ju dokáže dynamicky vytvárať jej kópie. Prvým vstupom *inlet* je list dát s presným formátovaním popisujúcim granulu v danom poradí:

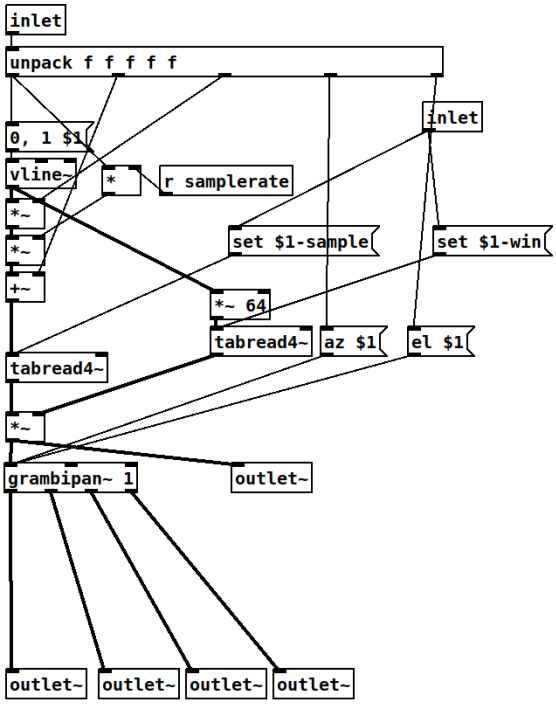
1. *grainsize* – veľkosť prehrávaného zrna (ms)
2. *position* – začiatočná pozícia prehrávaného zrna (sample)
3. *pitch* – výška tonu prehrávaného zrna (1 – originálna výška)
4. *azimut* – uhol prehrávaného zrna v stereobáze od 0° do 360° v protismere hodinových ručičiek (0 – 1)
5. *elevation* – zdvih prehrávaného zrna pri ambisonic dekódovaní (0 – 1)

Druhým vstupom je zadanie názvu tabuliek z vyššej úrovne patchu, z ktorých má danú granulu s danými parametrami prehrávať. Výhodou takéhoto zadávania je vytváranie viacerých unikátnych kópii vo vyšších vrstvách, keďže Pd má len globálne premenné. Po príchode listu dát na vstup sú následne rozbalené pomocou objektu *unpack* a rozložené do ďalších častí prehrávacieho algoritmu.

Ako prvé sa začne generovať signálová rampa pomocou objektu *vline~*. Rampa má rozsah hodnôt od 0 po 1 s argumentom *grainsize*, čo je čas v milisekundách, za ktorý má prejsť daný rozsah. Následne je táto rampa násobená parametrom *pitch*, ktorý mení jej rozsah bez zmeny jej času, čo ma za výsledok zmenu jej proporcie a teda výšky tónu. Ďalej je táto rampa zväčšená na veľkosť dĺžky zrna v *samploch*. Následne je priradený offset od začiatočného bodu prehrávania pričítaním premennej *position*.

Takto upravená signálová rampa je privedená na vstup objektu *tabread4~*, ktorý prehráva z udanej tabuľky hodnoty zapísaného signálu so 4-pólovou interpoláciou na výstupe. Tento výstupný signál je násobený signálom z ďalšej tabuľky, ktorá je čítaná súbežne. V nej je zapísaná priebeh obálky daného zrna (4.1.6). Amplitúdovo upravený signál je vyvedený na signálový výstup *outlet~*, pre použitie v sekcii s konvolúciou (4.1.4).

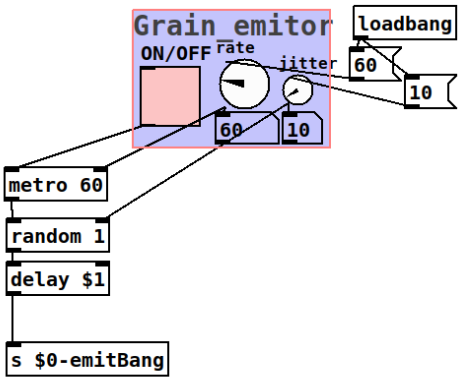
Ďalšia časť pokračuje do objektu *grambipan~*, ktorý z mono signálu na jeho vstupe a argumentov o jeho pozícii (*azimut* a *elevation*) vypočíta na svojich štyroch výstupoch signál s formátovaním *Ambisonic B-format.* Tento signál je smerovaný na signálové výstupy *outlet~,* aby mohol byť dekódovaný vo vyšších vrstvách patchu. Takýmto spôsobom ma každá prehraná granula svoju vlastnú veľkosť, offset, výšku tónu a polohu v priestore.



Obr. 4-10 abstrakcia grain

4.1.10 pd grain emitor

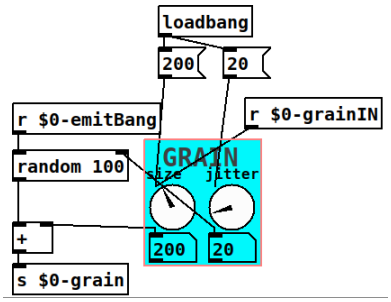
Je to subpatch na vyvolávanie časovaných udalostí (*bang*) pre synchronizované generovania jednotlivých parametrov a spúšťania prehrávania granúl. Používa objekt *metro,* ktorý generuje na svojom výstupe udalosti *bang* v zadanom časovom intervale. Tento interval zadávame objektom *mknob* s názvom *rate* a je určený ako interval v milisekundách. Parameter *jitter* určuje maximálnu možnú výchylku náhodnosti oneskorenia výstupnej udalosti, taktiež zadávaná v milisekundách. Výstupná udalosť je zapísaná do globálnej premennej *$0-emitBang* pomocou, ktorej je časované prehrávanie a ostatné zmeny jednotlivých parametrov subpatchov.



Obr. 4-11 pd grainEmitor

4.1.11 pd grain size

Tento subpatch (Obr. 4-12) slúži na užívateľské zadávanie veľkosti granúl. Objektom *mknob* s názvom *size* užívateľ zadá veľkosť granule v milisekundách. Druhý objekt *mknob* s názvom *jitter* určuje veľkosť náhodne generovanej odchýlky tejto veľkosti. Tieto údaje sú navzájom sčítané príchodom udalosti *bang* v premennej *$0-emitBang*. Následne sú poslané premennou *$0-grain* ďalej do ostatných subpatchov projektu.



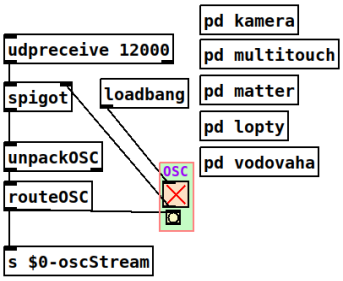
Obr. 4-12 pd grainSize

4.1.12 pd oscStream

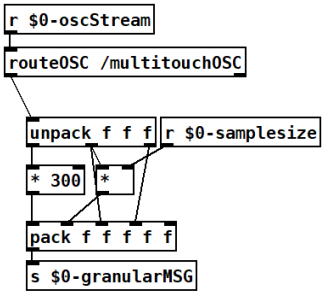
Objektom *udpraceive* s argumentom čísla portu zachytávame sieťovú komunikáciu v lokálnej sieti na danom porte. V našom prípade je táto komunikácia uskutočňovaná na porte 12000 na serveri *localhost*.

Stream dát pokračuje do objektu *unpackOSC* z knižnice *mrpeach*, ktorým sú selektované a na výstup prepustené len správy s formátovaním OSC. Ostatné sú zahodené. Následne sú triedené podľa ich prefixu (2.6.2) objektom *routeOSC*, taktiež z knižnice *mrpeach*. Takto selektované dáta sú posielané do premennej *$0-oscStream*, ktorú používajú ďalšie subpatche spracúvajúce dáta z jednotlivých GUI

webových skriptov, na ktorých základe sú posielané správy pre prehrávanie granúl (Obr. 4-14). V týchto subpatchoch sa priradzujú získané dáta jednotlivým parametrom do správ pre tvorbu granúl (4.1.3). Získavané dáta sú opísané v ďalších kapitolách (4.2.2).



Obr. 4-13 pd oscStream



Obr. 4-14 pd multitouch

4.2 Vzdialené ovládanie

Vyššie opísané funkčné prvky slúžili na ovládanie parametrov užívateľom v rámci prostredia Pd a boli viac-menej klasickým ovládaním granulárnej syntézy. Čím však vyniká tento nástroj je to, že tieto jednotlivé parametre môžu byť ovládané vzdialene, viacero zariadeniami súbežne, po sieti, v ktorej je daný počítač s patchom pripojený. Počítač s patchom teda slúži ako server a spracováva jednotlivé vstupy a premieňa ich na zvukový výstup. Takto je oddelený zvukový zdroj od vstupu a parametrov pre jeho tvorbu.

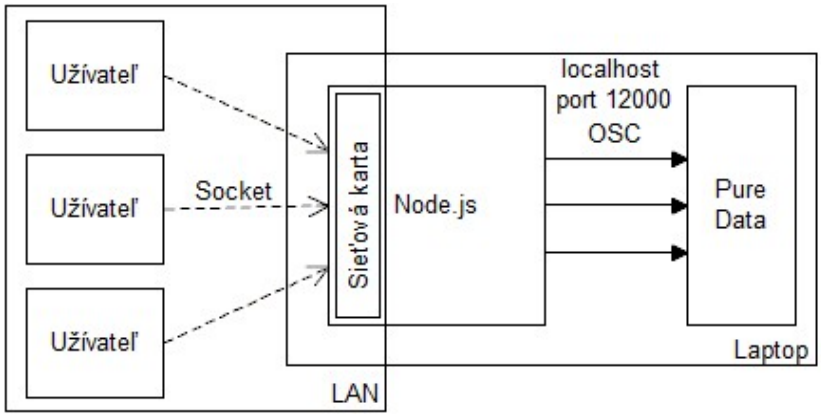
Pri použití tohto oddelenia sa naskytujú nespočetné možnosti kreatívneho a expresívneho ovládania týchto známych parametrov syntézy. Ďalšou skvelou výhodou je, že ovládanie nie je obmedzené na jedného užívateľa, ale poskytuje priestor pre kolaboráciu zariadení v reálnom čase. Každé zariadenie generuje dáta samo na svojom procesore. Každá prijatá správa je menená na zvuk. V tejto sekcii

opíšem akým spôsobom funguje externé ovládanie a rozoberiem jednotlivé ovládacie skripty.

4.2.1 Server

Nutnosťou pre viac užívateľskú komunikáciu je centralizované prijímanie dát od užívateľov pomocou servera. Server slúži ako stredný blok komunikácie medzi webovými skriptami ovládania a zvukovým zdrojom v Pure Data a zároveň distribuuje ovládacie skripty. Je síce možné komunikovať priamo zo zariadenia po sieti s Pd patchom, no táto cesta vyžaduje natívnu aplikáciu na danom zariadení. Lokálna sieť disponuje nízkou latenciou a priamym komunikačným spojením. Webové stránky s podporou Java Scriptu dokáže otvoriť drvivá väčšina dnešných moderných zariadení.

Na ukážke topológie (Obr. 4-15) vidno komunikačnú hierarchiu typu klient-server. Server je realizovaný pomocou technológie Node.js (2.7.3). Užívateľ pripojený na rovnakú sieť ako zariadenie, ktoré ju zdieľa, dokáže sa pomocou IP adresy a čísla portu pripojiť priamo na serverom zdieľanú webstránku. Z nej si dokáže vybrať ľubovoľný skript a používať ho. Skript komunikuje so serverom pomocou technológie Web Socket (2.7.5) na porte 3000. Server následne tieto správy formátuje na OSC protokol a tieto dáta posiela do Pd po lokálnej sieti na porte 12000, kde sú premenené na zvuk.



Obr. 4-15 Topológia

Ukážka kódu (Ukážka kódu 1) znázorňuje vytvorenie komunikačného serveru pomocou systému Node.js. Prv je vykonané načítanie realtime modulu *express* (2.7.4) a následne priradenie portu, na ktorom má komunikovať po lokálnej sieti. Pomocou príkazu *app.use(express.static(public))* je verejne zdieľaná zložka s názvom *public*.

V tejto zložke sa nachádzajú všetky verejne prístupne webové skripty a knižnice. Ďalej nasleduje vytvorenie modulu *socket.io* pre komunikáciu pomocou Web Socketov a jeho priradenie k danému vytvorenému serveru na porte 3000. Na záver je vytvorená komunikácia protokolom OSC na porte 12000 pomocou modulu *node-osc*.

```
var express = require('express'); // express node framework
var app = express();
var server = app.listen(3000); // priradenie komunikačného  portu 3000

app.use(express.static('public'));  //zdieľa verejnej zložky
var socket = require('socket.io'); // zavolanie modulu socket.io
var io = socket(server);            // priradenie modulu socket.io na server

io.sockets.on('connection',newConnection);

var osc = require('node-osc');   // zavolanie modulu node-osc
var client = new osc.Client('127.0.0.1', 12000); // priradenie portu modulu
```

Ukážka kódu 1 Nastavenie serveru

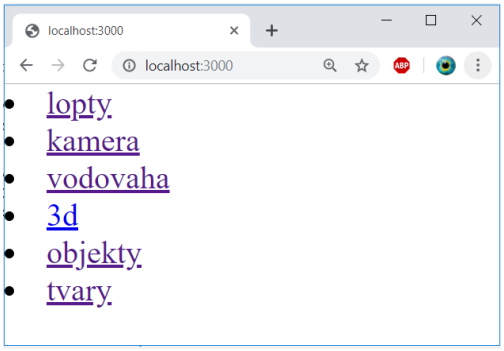
Posielanie dát zo skriptu na server bude popísaná v nasledujúcich podkapitolách. Teraz sa zameriame na spracovanie udalostí po prijatí správy (Ukážka kódu 2). Po pripojení nového užívateľa je spustená udalosť *newConnection* na danom *socket*e. Vytváram tam takzvaný *eventListener*, ktorý čaká na vykonanie udalosti s daným názvom. V našom prípade je to príchod správy s názvom *multitouch*, po ktorej príchode je vykonaná *callback* funkcia s názvom *multitouchMsg*. Úlohou tejto funkcie je preposlať dáta prichádzajúce v správe danej udalosti ako OSC správu na definovanom porte s definovaným názvom. Táto správa bude poslaná do Pd a tam rozbalená a spracovaná. Takéto *eventListenery* je potrebné vytvoriť pre každú prichádzajúcu správu od skriptov.

```
function newConnection(socket){       //eventListener pre nového užívateľa
  socket.on('multitouch',multitouchMsg); //udalosť s názvom správy
    function multitouchMsg(data){ //definícia callback funkcie
          client.send('/multitouchOSC', data);
    }
}
```

Ukážka kódu 2 Nastavenie event listenera pre správu

4.2.2 Web

Po zapnutí server verejne zdieľa stránku zložku obsahujúcu hlavnú web stránku *index.html*, všetky potrebné knižnice a jednotlivé ovládacie skripty. Preto nie je potrebné pripojenie k internetu na ovládanie zvuku. Všetky knižnice potrebné na funkciu sú uložené a zdieľané na serveri. Po presmerovaní na adresu zariadenia, na ktorom beží server je zobrazená hlavná web stránka *index.html* (Obr. 4-16). Táto stránka obsahuje hypertextové odkazy na jednotlivé JavaScriptové ovládacie skripty. Skripty sú písané prevažne tak, aby sa prispôsobili zariadeniu, na ktorom sú spustené bez veľkých obmedzení funkcie a ovládania.



Obr. 4-16 Index.html

Každý z týchto skriptov ovláda parametre tvorby zvuku a jeho priestorového rozloženia iným spôsobom. Jednotlivé skripty budú rozoberané v kapitolách nižšie. Každý zo skriptov generuje dáta rôznymi spôsobmi, tie vytvárajú Web Socket správy spracované na serveri (4.2.1). Výhodou tohto systému je to, že každé zariadenie používa svoj procesor na generovanie dát a tým nezaťažuje procesorový čas serveru, ktorý ho využíva na spracovanie dát a tvorbu zvuku.

4.2.2.1 Skript Lopty

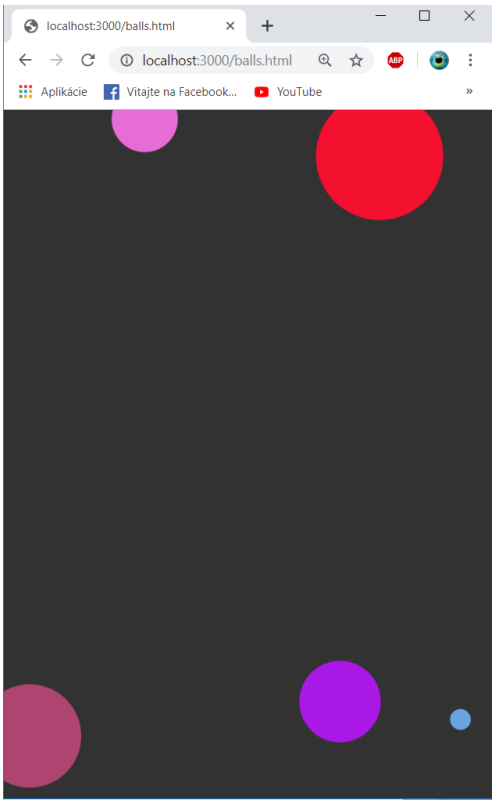
Tento interaktívny skript je vytvorený pomocou p5.js (2.7.1) Po načítaní je zobrazovaný na celú obrazovku zariadenia, na ktorom beží. Skript (Obr. 4-17) po dotyku prsta alebo kliku myšou vytvorí guličku s náhodnou veľkosťou, farbou, smerom a rýchlosťou jej pohybu. Táto gulička putuje po obrazovke a odráža sa od jej stien. Každým nárazom guličky o stenu je poslaná správa na server. Táto správa je formátovaná nasledovne:

- **x** – pozícia nárazu na ose x

- **y** – pozícia nárazu na ose y
- **d** – veľkosť danej guličky
- **uhol** – uhol nárazu od stredu obrazovky

Jednotlivé hodnoty sú normalizované z ich celého rozsahu na rozsah 0-1 pre ľahšie spracovanie v Pd. Takýchto guličiek je možné vytvoriť celkovo 5. Odstránenie guličky je možné kliknutím na danú existujúcu guličku. Ďalšou možnosťou je dvojklik na ľubovoľné miesto, tým sa odstráni posledná pridaná gulička. Potiahnutím v ľubovoľnom smere prstom alebo myšou sa zmení smer a rýchlosť jednotlivých guličiek. Rýchlosť potiahnutia ovplyvňuje silu, ktorá bude pridaná daným guličkám. Ich smer sa mení aj naklonením zariadenia s gyroskopom. Ten zmení smer guličiek v smere náklonu zariadenia.

Výstupy tohto skriptu sú v Pd premenené na ovládanie týchto parametrov, **x** - ovláda pozíciu zrna, **y** - ovláda výšku tónu, **uhol** - pozíciu panorámy a **d** - ovláda veľkosť.



Obr. 4-17 skript Lopty

Skrytou vymoženosťou tohto skriptu je vytvorenie náhleho hustého asynchrónneho oblaku tým, že zariadenie dostatočne dlho nakláňame jedným smerom. Po čase všetky guličky padnú na jednu stranu a začnú prepadávať cez stenu

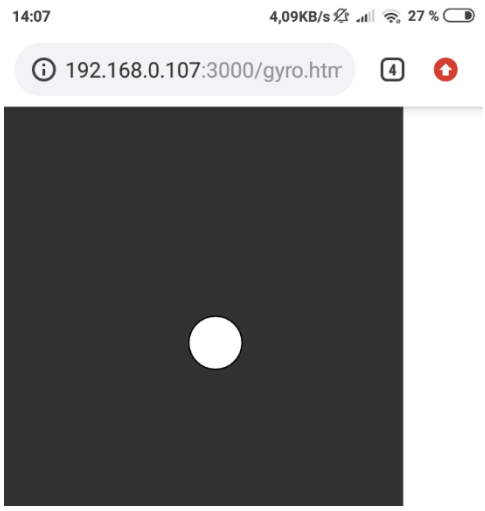
displeja, čo vyvolá mnohonásobné posielanie správ do Pd. Krásou tohto oblaku je, že ho nie je možné vytvoriť ihneď, ale treba naň počkať. Jemným nakláňaním ho je možné posúvať po stenách a tým meniť jeho celkový zvuk.

4.2.2.2 Skript Vodováha

Tento interaktívny skript je taktiež vytvorený pomocou p5.js (2.7.1) a slúži na priestorovú panorámu konvolúcie. Nakláňaním zariadenia s gyroskopom sa gulička plynule pohybuje po obrazovke (Obr. 4-18) a posiela správy o svojej polohe pri každej zmene polohy zariadenia. Tieto dáta sú formátované a použité nasledovne :

- Uhol – je uhlom, ktorý zviera gulička so stredom obrazovky a v Pd je použitý ako ovládanie horizontálnej panorámy
- Zdvih – vertikálna pozícia zariadenia z gyroskopu, je v Pd použitá ako zdvih vertikálnej roviny konvolúcie

Výhodou tohto skriptu je jeho plynulosť. Konvolučný reverb neznie veľmi dobre, ak sa jeho pozícia vo zvukovom priestore mení skokovito a náhle. Tento skript ovláda jeho pozíciu jemne a bez preskokov. Tento skript by nemalo používať viacero užívateľov naraz, pretože správy budú spôsobovať nechcenú skokovitosť.



Obr. 4-18 skript Vodováha

4.2.2.3 Skript kamera

Ide o experimentálny skript, ktorý bol vytvorený pomocou p5.js (2.7.1). Funkcia spočíva v použití kamery zariadenia a analyzovaní farieb. Po kliknutí na ľubovoľný

bod obrazu je na danom mieste analyzovaná farba pixelu. Následne uložená do premennej a jeho algoritmus hľadá na celom obraze farbu, ktorá spadá pod kritéria podobnosti. Na mieste nálezu vykreslí biely obdĺžnik (Obr. 4-19), ktorého veľkosť sa mení podľa toho, koľko podobných farieb v blízkosti našiel. Dvojklikom je možné vypnutie vyhľadávania farieb.

Každý vykreslený obdĺžnik posiela správy o svojej pozícii X,Y a o svojej veľkosti. Tieto dáta sú v Pd spracované ako veľkosť granule, pozícia granule, výška tónu granule a jeho priestorové rozloženie. Výsledný zvuk záleží od kvality kamery a dobrej svetelnosti. Zvuk rázne ovplyvní aj výber vyhľadávanej farby. Ak je farba na obrazovke v menšom zastúpení, výsledným zvukom sú krátke občasne nabiehajúce ústrižky. Kdežto pri vybraní farby, ktorá sa nachádza na obraze v hojnom zastúpení sa radikálne mení hustota zvuku na mnohonásobne sa prekrývajúci sa oblak granúl. Zaujímavé je taktiež pracovať s expozíciou kamery, ktorá spôsobí posun granúl alebo zmenu hustoty oblaku. Ďalšia zaujímavosť je použitie svetla ako najsvetlejšieho bodu obrazu a tým priamo kontrolovať malý prúd granúl po priestore.

Skript nie je funkčný na niektorých zariadeniach v sieti, pretože pre použitie kamery na sieti je v niektorých prípadoch nutné zabezpečenie *https*, ktoré tento server zatiaľ nemá.

Obr. 4-19 skript kamera

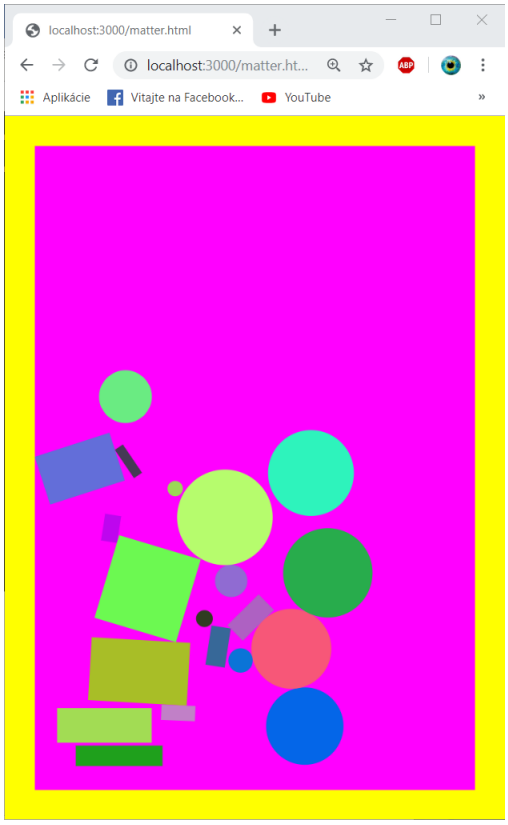
4.2.2.4 Skript Objekty

Veľmi expresívny skript s názvom objekty bol vytvorený pomocou p5.js (2.7.1) na vizualizáciu objektov a Matter.js (2.7.2) pre počítanie 2D fyziky týchto objektov.

Skript (Obr. 4-20) generuje množstvo objektov s náhodnou veľkosťou, farbou a tvarom. Sú vložené do 2D sveta, ktorého gravitácia sa mení naklonením zariadenia. Ak je zariadenie v stabilnej polohe na stole, gravitácia je nulová. Avšak zmenou polohy sa gravitácia nakláňa v smere zmeny a objekty sa začnú hýbať a vrážať do seba. Gravitácia reaguje taktiež na akcelerometer zariadenia. Čím väčšia akcelerácia zmeny polohy, tým rýchlejšia a náhlejšie zmena gravitácie. Zmenu gravitácie je možné docieliť stláčaním šípkových kláves na počítačoch.

Objekty svojím narážaním vyvolávajú udalosti tzv. *collisionEvent*. Keďže na výpočet fyziky je použitá knižnica *Matter.js* (2.7.2), udalosti nárazov jednotlivých párov objektov obsahujú mnoho dát o danej kolízii. V tejto verzii skriptu sa pri každej kolízii pošle správa o veľkosti kolidujúceho objektu, *gamma* naklonení zariadenia, sile daného nárazu, uhle zvieraného od stredu displeja s centrom kolízie, a alfa naklonení zariadenia.

S každým objektom je možné interagovať dotykom. Posúvať ho, vytvárať kolízie s ostatnými a podobne. Tieto dáta sú posielané a v Pd sú premenené na ostré zvuky nárazov. Veľkosti kolidujúceho objektu je priradené ovládanie veľkosti granule, náklon zariadenia mení pozíciu granule v súbore, sila kolízie určuje výšku tónu, uhol ovláda pozíciu prehrania granule v priestore a zdvih (*alfa* náklon) zariadenia určuje zdvih pozície granule v horizontálnej ose. Výsledkom sú zvuky meniace svoju ostrosť agresivitou pohybovania zariadením. Vznikajú dezorganizované nepredvídavé a náhle sa meniace nárazy zvukov. Tento skript exceluje ak je v patchy načítaný zvuk úderov pohárov, železa alebo iný širokospektrálny, husto umiestnený zvukový záznam. Zaujímavou kombináciu v tomto prípade je načítanie podobného charakteru zvuku do konvolučného algoritmu, kde sa tieto spektrá násobia a vytvárajú ešte nepredvídavejšie ústrižky z toho, čo kedysi boli reálne zvuky.

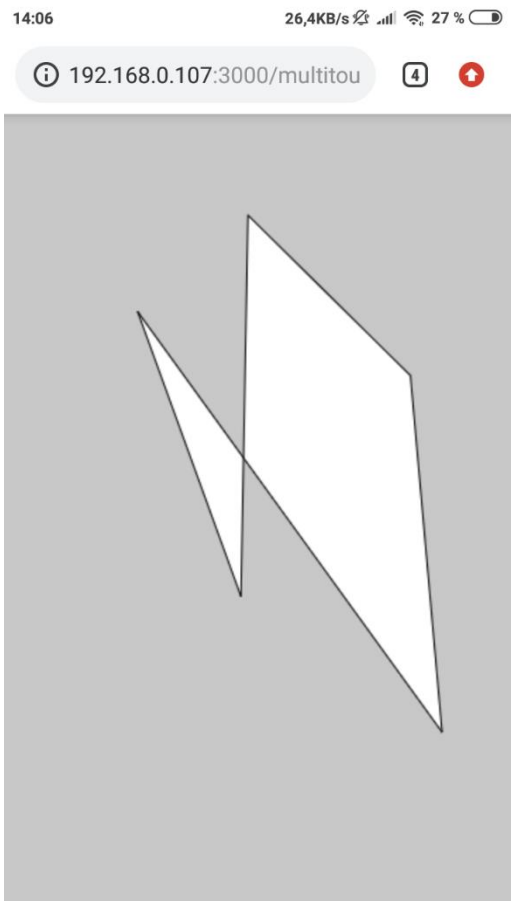


Obr. 4-20 skript Objekty

4.2.3 Skript dotyky

Ďalší veľmi expresívny skript, s názvom *multitouch*, bol vytvorený pomocou p5.js (2.7.1). Ako jeho názov hovorí, ide o spracovanie viacdotykových vstupov na displejoch zariadení . Funkciou skriptu (Obr. 4-21) je posielanie dát o jednotlivých dotykoch na displeji ako sú ich pozícia x, y a uhol zvierajúci so stredom. Dotyky sú vykreslené ako body v mnohouholníku, ktorý je uzavretý a vyplnený bielou farbou. Tieto dáta sú posielané pre každý dotyk zvlášť pri zmene jeho pozície. V Pd sú spracované nasledovne: pozícia X ovláda veľkosť granule, pozícia Y ovláda pozíciu granule, výšku tónu a každá z týchto granúl je posadená do panorámy podľa zvieraného uhlu daného dotyku so stredom displeja.

Zaujímavé je, že správa o dotykoch je posielaná iba pri zmene polohy a nie pri obyčajnom kliknutí. Takto je potrebné neustále hýbať jednotlivými prstami, aby boli dotyky zaznamenané. Výsledný zvuk predstavuje priestorový oblak jemne rozladených granúl pod každým z prstov, ktorý sa dotýka obrazovky. Je vhodný na vytváranie dlhých nôt a dronov ambientného charakteru. Počet dotykov spracovaných skriptom je obmedzený zariadením a nie skriptom samotným.



Obr. 4-21 skript Dotyky

4.3 Zvukové vlastnosti

Výsledný nástroj disponuje radom možností ako tvoriť komplexné zvukové zložky s jednoduchosťou a hravosťou. Napríklad aj takto sa dá vytvoriť komplexný priebeh granulácie. Po načítaní alebo nahraní zvukového súboru je možné transformovať tento zvuk na plochu pomocou spustenia *emitora* granúl s nízkym nastavením parametra rate, parametrom *grainsize* na veľkosť približne 100 ms. Posúvaním pozície získavame hustý mrak zŕn z okolia zadanej polohy v súbore. Parametrom *pan jitt* ich môžeme náhodne rozhadzovať po priestore. Na kontrast k tomuto hustému zvuku, pripojením skriptu počas prehrávania mraku a následným pohybom zariadenia, na ktorom je spustený, vznikajú agresívne asynchrónne nárazy zvukov s ostrými priestorovo rozloženými zrnami. Zapnutím sekcie *conv* a načítaním IR z výstupu granulárneho syntezátora toto výsledné komplexné spektrum obohatíme o konvolúciu s ním samým.

Po vypnutí skriptu objekty a *emitora* ostáva počuť sekundu dlhý dozvuk po konvolúcii. Zapnutím skriptu *multitouch* a jemnými krátkymi ťahmi po obrazovke spúšťame krátke zhluky granúl v priestore, meniace svoju výšku podľa polohy dotyku. Tieto granule rozoznievanú komplexný konvolučný dozvuk z predošlého výstupu granulátora, ktorý znie a putuje po priestore nezávisle ovládaním skriptu vodováha. Ten priestorovo ovládame gyroskopom v skripte vodováha.

Takto behom niekoľkých sekúnd vznikol hustý mrak s ostrými výškovými nárazmi, narastal o konvolučný dozvuk. Následne opadol a prešiel do krátkych zhlukov atonálnych granúl obsadený tichom medzi jednotlivými dotykmi. Väčšiu komplexnosť je možné súbežným používaním viacerých skriptov viacerými užívateľmi.

5. ZÁVER

Cieľom tejto práce bolo vytvoriť experimentálny softwarový hudobný nástroj vyznačujúci sa neobvyklou kvalitou zvuku a ovládania. Predložená práca preto obsahuje základné teoretické poznatky nutné k jeho riešeniu. Tam boli popísané jednotlivé typy syntéz, programovacie prostredie a technológie, ktoré boli použité.

Výsledkom je hudobný nástroj, ktorý je postavený na mnohých voľne prístupných technológiách dnešnej doby. Nástroj so schopnosťou expresívne generovať širokú škálu priestorovo rozloženého zvuku za pomoci granulárnej syntézy a konvolúcie. Zvuky, ktoré tento nástroj vytvára sa pohybujú v rozmedziach od najkratších zvukových ústrižkov, o dĺžke jednotiek milisekúnd s nepravidelným a riedkym prehrávaním a mono výstupom, až po monumentálne široké, mnohonásobne prekrývajúce sa digitálne mraky zvukových granúl distribuovaných v priestore sterea, ako aj mnohokanálovej reprodukcie formátovania B-Format s konvolučným dozvukom ľubovoľnej impulznej odozvy. Celá táto škála zvukov je prístupná množstvu užívateľov pripojených na lokálny server z ľubovoľného zariadenia ako je napríklad smartphone alebo laptop, formou interaktívnych webových aplikácii. To všetko je prístupné v reálnom čase. Hravé ovládanie, sieťová kolaborácia, nezávislosť na platforme zariadenia, systém ovládania a presného kódovania každej jednej granule v mnohokanálovom reprodukčnom systéme sú výslednými funkciami nástroja vytvoreného v tejto práci. Systém sa snaží podnecovať kreativitu do komponovania nových zvukov oddelením zvukovej zložky od ovládacieho systému. Pri tvorbe tejto práce som sa naučil mnohé informácie o tvorbe zvuku, priestorovom audiu a webe. Stále je čo vylepšovať a do budúcna je možností mnoho. Sprístupniť používanie viacerých zvukových zdrojov z mnohých tabuliek súbežne, generovanie interaktívneho vizuálneho výstupu systémom a ďalšie. Vytvorenie ďalších ovládacích webových aplikácii alebo premenenie systému v natívnu aplikáciu pre Android či iOS. Tento systém poskytuje škálovateľnosť, na ktorej je možné tieto vylepšenia realizovať.

Literatura

[1] RUSS, Martin. *Sound synthesis and sampling*. 2nd ed. Boston: Elsevier/Focal Press, c2004. ISBN 0240516923.

[2] ROADS, Curtis. *Microsound*. Cambridge, Mass.: MIT Press, c2001. ISBN 0262182157.

[3] KURC, David. *Snímání a zpracování akustických signálů technologií B Format: Acoustic Signal Recording and Processing Using B-Format Technology*. Brno: Vysoké učení technické, Fakulta elektrotechniky a komunikačních technologií, 2009.

[4] PUCKETTE, Miller. *The theory and technique of electronic music*. Hackensack, NJ: World Scientific Publishing Co., c2007. ISBN 9812700773.

[5] Oficiálna stránka PureData [online]. [cit. 2018-12-14]. Dostupné z: https://puredata.info/

[6] Oficiálna stránka protokolu OSC [online]. [cit. 2018-12-14]. Dostupné z: http://opensoundcontrol.org

[7] Oficiálna stránka programovacieho jazyka Processing [online]. [cit. 2018-12-14]. Dostupné z: https://processing.org/

[8] SHIFFMAN, Daniel. *Learning processing: a beginner's guide to programming images, animation, and interaction*. Second edition. Amsterdam: Elsevier/Morgan Kaufmann, [2015].

[9] Oficiálna stránka p5.js [online]. [cit. 2018-12-14]. Dostupné z: https://p5js.org/

[10] Oficiálna stránka frameworku Node.js [online]. [cit. 2018-12-14]. Dostupné z: https://nodejs.org/en/about/

[11] RAI, Rohit. *Socket.IO Real-time Web Application Development*. 1. Packt Publishing, 2013. ISBN 9781782160786.

[12] TICHÝ, Vladimír. *Digitální zvukový efekt typu reverb využívající konvoluci signálu s impulsní charakteristikou poslechového prostoru*. Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií, 2009.

Zoznam príloh

Příloha 1 - Zdrojový kód skriptu Dotyky

```
var colors;
var grains = [];
var socket;
var angle ;

function setup() {
  createCanvas(displayWidth, displayHeight);
  background(200);
  pixelDensity(1);
  socket = io.connect(':3000');
}

function draw() {
  background(200);
  beginShape();
  fill(255);
  for (var i = 0; i < touches.length; i++) {
    vertex(touches[i].x, touches[i].y);
  }
  endShape(CLOSE);
}

function touchMoved(event) {
  var touch = event.changedTouches;
  var dlzka = touch.length;
  var x, y , mapX, mapY;
  var uhol;
  let i = 0;
  for(i; i < dlzka; i ++){

    x = (touch[i]).clientX;
    y = (touch[i]).clientY;
    mapX = map(x,0,displayWidth,0,1);
    mapY = map(y,0,displayHeight,0,1);
    uhol = vypocetUhlu(x,y);
    if(i == 0)socket.emit('multitouchLast',[mapX, mapY,uhol]);
    socket.emit('multitouch',[mapX, mapY,uhol]);
  }
  return false;
}
```

```
function vypocetUhlu(x,y){
  //vypocet uhlu dotyku od stredu a prepocet 0-1 proti hodinovym rucickam&
  translate(displayWidth/2, displayHeight/2);
  var angle = atan2((displayHeight/2)-y,(displayWidth/2)-x);
  return map(((degrees(angle)+270)%360),0,360,1,0);
}

function grain(r){
  this.x = 0;
  this.y = 0;
  this.r = r;
  this.col = [random(255),random(255),random(255)];
  print(this.x,this.y);grains.push(new grain(random(10,100)));
  this.show = function(x,y){
    this.x = x ;
    this.y = y;
    push();
    fill(this.col);
    ellipse(x,y,this.r,this.r);
    pop();
  }
}
```

Příloha 2 - Zdrojový kód skriptu Objekty

```
var Engine = Matter.Engine,
World = Matter.World,
MouseConstraint = Matter.MouseConstraint,
Mouse = Matter.Mouse,
Events = Matter.Events;
Bodies = Matter.Bodies;

var engine;
var objekty = [];
var world;
var mConstraint;

var steny = [];
var canvas;

var alpha,beta,gamma;
var w,h;
var r = 255 ,g = 0 ,b = 255;
var zdvih = 0;
var naklon = 0;
var maxPlocha = 0;
var col;
```

```
function setup(){
  w = windowWidth;
  h = windowHeight;
  canvas = createCanvas(w,h);
  engine = Engine.create();
  world = engine.world;
  socket = io.connect(':3000');
  col = color(255,0,255);

  //pridanie stien
  steny.push(new Stena(0,0,w*2,30));//horna
  steny.push(new Stena(0,h/2,30,h));//lava
  steny.push(new Stena(w,h/2,30,h));//prava
  steny.push(new Stena(w/2,h,w,30));//dolna

  //pridanie mysi
  var canvasmouse = Mouse.create(canvas.elt);
  canvasmouse.pixelRatio = pixelDensity();
  mConstraint = MouseConstraint.create(engine,{
    mouse: canvasmouse
  });

  World.add(world,steny);
  World.add(world,mConstraint);

// hlavny event handler
  Events.on(engine, 'collisionStart', function(event){
      let pairs = event.pairs;
      for (let i = 0; i < pairs.length; i++){

          let bodA = pairs[i].bodyA;
          let bodB = pairs[i].bodyB;
          let sila = Math.abs(bodB.velocity.x *bodB.velocity.y);

          if(sila > 0.2){
            if(bodA.label == 'Stena'){
              var uhol = vypocetUhlu(bodB.position.x,bodA.position.y);
              var velkost = map(bodB.area,0,maxPlocha,0,500);
            }
            else{
              var uhol = vypocetUhlu(bodA.position.x,bodA.position.y);
              var velkost = map(bodA.area,0,maxPlocha,0,500);
            }
            socket.emit('matter', [ velkost,
naklon,map(sila,0.05,100,0.8,5),uhol,zdvih]);
          }
        }
```

```
  });


//pridavanie kociek
  for (var i = 0; i < 10 ; i++){
    var sirka = random(w/70,w/5);
    var vyska = random(w/70,w/5);
    //var plocha = sirka * vyska;
    objekty.push(new Box(random(w/4,w/2),random(h/4,h/2),sirka,vyska));
    if(objekty[i].body.area >= maxPlocha) maxPlocha = objekty[i].body.area;
  }
  //pridavanie lopt
  for (var i = 0; i < 10 ; i++){
    var sirka = random(w/70,w/5);
    var vyska = random(w/70,w/5);
    objekty.push(new
Lopta(random(w/4,w/2),random(h/4,h/2),random(w/70,w/5)));
    if(objekty[i].body.area >= maxPlocha) maxPlocha = objekty[i].body.area;
  }
}

function draw(){
  Engine.update(engine);
  background(col);
  for(var i = 0 ; i < steny.length; i++){
    steny[i].show();
  }
  for(var i = 0 ; i < objekty.length; i++){
    objekty[i].show();
  }
}

function zoom(){
  print(event);
}

function deviceMoved() {
  //akceleracia objektov
  r = map(accelerationX, -90, 90, -20, 20);
  g = map(accelerationY, -90, 90, -20, 20);
  world.gravity.y += g;
  world.gravity.x -= r;

}

//zmena gravitacie pre pc
function keyPressed(){
  if(keyCode === LEFT_ARROW){
```

```
    world.gravity.x -= 0.1;
  }
  if(keyCode === RIGHT_ARROW){
    world.gravity.x += 0.1;
  }
  if(keyCode === UP_ARROW){
    world.gravity.y -= 0.1;
  }
  if(keyCode === DOWN_ARROW){
    world.gravity.y += 0.1;
  }
}

window.addEventListener('deviceorientation', function(e)
{
  zdvih = map(e.beta, -90 , 90 , 0 ,1);
  naklon = map(e.gamma, -90 , 90 , 0 ,1)%1;
  world.gravity.y = e.beta/30;
  world.gravity.x = e.gamma/30;
});

function vypocetUhlu(x,y){
  //vypocet uhlu dotyku od stredu a prepocet 0-1 proti hodinovym rucickam pre
Pd
  push();
  translate(displayWidth/2, displayHeight/2);
  var angle = atan2((displayHeight/2)-y,(displayWidth/2)-x);
  pop();
  return map(((degrees(angle)+270)%360),0,360,1,0);

}
```